AD-A264 375

Technical Report
CMU/SEI-92-TR-32
ESC-TR-92-032

Carnegie Mellon University
Software Engineering Institute

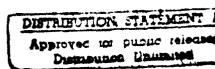# Performance and Ada Style for the AN/BSY-2 Submarine Combat System

Neal Altman
Patrick Donohoe

December 1992

DTIC
ELECTE
MAY 1 8 1993
S
C
D

Performance and Ada Style for the AN/BSY-2 Submarine Combat System

Neal Altman
Patrick Donohoe

Real-Time Embedded Systems Testbed (REST) Project

D⠶⠶⠶   ⠶ B

Accesion For		
NTIS CRA&I	☑	
DTIC TAB	☐	
U. announced	☐	
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESC/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official
DoD position. It is published in the interest of scientific and technical
information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

Table of Contents

List of Figures

List of Tables

Performance and Ada Style for the AN/BSY-2 Submarine Combat System

Abstract: The performance of programs prepared with the Verdix Ada Development System (VADS) was measured and analyzed for programmers preparing a large Ada system. Using standard Ada benchmark suites (ACEC, AES and PIWG) and a representative Motorola 68030 target system as a source of data, questions were posed and answered about programming alternatives, based on the measured performance of the compiler. The questions included in the report were extracted from a much larger set selected from an analysis of the BSY-2 Style Guide and augmented with additional questions suggested by SEI experience. The derivation of the questions and the template for the performance analysis sections are presented as appendices.

1   Introduction

The U. S. Navy's AN/BSY-2 Submarine Combat System will use the Ada programming language for the majority of its software. The coding style used in preparing this software will affect the readability, testability and maintainability of the code during the software life cycle. Programming style can also affect the performance of the software in operation. The emphasis of this report is to describe the effect of Ada coding style on the execution performance of Ada programs. Impact on memory utilization is a secondary concern but is mentioned when appropriate.

This report has been organized as a series of questions. Each question provides information about programming choices. In developing a software system, implementors make a large number of choices about the design and coding of software. Because software is inherently flexible and Ada provides a large number of alternatives to the programmer, the correct or most efficient choice among the alternatives is not always known.

This report emphasizes the performance consequences of Ada coding choices, rather than characterizing Ada's performance or comparing Ada to other computer programming languages. It abstracts relevant information from Ada benchmark suites and presents those data and a series of conclusions. Performance is, of course, only one aspect to be considered in making programming choices. The code's consistency, clarity, maintainability, plus other factors, must also be considered. This report is intended to inform rather than dictate. In particular, the intent is to allow rational choices to be made with knowledge of the performance consequences, rather than suggesting that performance is or should be the overriding consideration. Often the differences in performance between alternative choices are shown to be negligible.

While the questions asked by this report are intended to be of general interest to Ada users, the conclusions are only applicable to the tested hardware and software configurations specified in the individual entries. The primary source of performance information is a configuration using the Verdix Ada compiler selected for BSY-2 and commercial-off-the-shelf hardware used for the initial prototype of the BSY-2 system. Where appropriate, conclusions are also drawn from other systems and the work of other government agencies. However, it should be emphasized that these results are specific recommendations for BSY-2 and are not universally applicable to all Ada systems or hardware platforms. As new

system hardware and software are deployed, all relevant tests must be re-executed and the results examined to ensure the recommendations remain current.

1.1 BSY-2 Performance Questions

This section presents the performance questions addressed in this report. Appendix A on page 181 describes how the list of questions was selected. Performance information for each question is provided in the body of the report (Performance Reports on page 3).

1.2 Performance Questions List

This list is organized alphabetically by topic.

1. **Arrays:** What are the performance characteristics of array aggregate assignments and corresponding loop constructs?

2. **Check Suppression:** How does performance change when checks are turned off?

3. **Data Location:** What is the performance effect of declaring data locally or outside the executing scope?

4. **Enumeration Types:** How does the performance of operations on objects of an enumeration type compare with the performance of an equivalent representation using strings or numeric values?

5. **Exceptions:** What are the performance consequences of providing exception handling capabilities?

6. **Generic Units:** What is the comparative performance of generic and nongeneric units?

7. **Inlining of Procedures:** What is the effect of inlining procedures and generic procedures?

8. **Logical Tests:** What are the performance tradeoffs between the **case** statement and **if** statement?

9. **Loop Efficiency:** Do different loop constructs vary in efficiency?

10. **Module Size:** Is the performance of a program divided into modules different from a monolithic design?

11. **Optimization Options:** What are the effects of different optimization levels?

12. **Precision:** What are the performance differences between single-precision and extended-precision numeric operations?

13. **Private Types:** Is there a difference in performance between operations on objects of a private type and objects of a visible type?

14. **Records:** What is the performance of the various methods for assigning values to record objects?

15. **Rendezvous:** What are the performance characteristics of the various kinds of task rendezvous?

2　Performance Reports

This section contains observations based on benchmark tests. Each section opens with a question about performance, style, and/or features, and an answer to the question plus the detailed data that support the answer. The questions are arranged alphabetically by the key subject.

2.1　Background

The performance reports were developed as a method of presenting interesting results from the substantial body of data generated by Ada benchmark test suites. Each performance report answers a specific question and uses data from a selected subset of benchmark tests that specifically address this question.

Several basic decisions were made about the development of the performance reports:

- The report would focus on choices available to the application programmer.
- The BSY-2 Style Guide would be the primary source of questions.
- Data from the standard Ada benchmark suites would be used, rather than customized tests.
- Limitations of the current benchmark suites (e.g., missing tests) would be pointed out as encountered.

2.1.1　BSY-2 Style Guide

The BSY-2 Style Guide is contained in Appendix I of the document "Software Standards and Procedures Manual for the AN/BSY-2 SUBMARINE COMBAT SYSTEM." It lays out coding standards for Ada programmers and is written to reflect commonly held standards of good programming practice. However, it does not address the performance of the resulting Ada code.

In addition to the issues raised by the BSY-2 Style Guide, the report authors decided to include additional questions to cover issues not raised in the style guide. In preparing these questions some consideration was given to including issues that were well addressed by the then current releases of the standard Ada benchmark suites.

Appendix A on page 181 lists the questions derived from the BSY-2 Style Guide.

2.1.2　Benchmark Suites

Performance data for the report were obtained by use of commonly available benchmark suites. The suites used for this report are all written in Ada:

PIWG A benchmark suite produced by a volunteer group, the Performance Issues Working Group of the Special Interest Group for Ada (SIGAda),

(a part of the Association for Computing Machinery or ACM). The PIWG test suite is compact, widely used and available without charge.

ACEC The Ada Compiler Evaluation Capability is a set of benchmark tests prepared for the Ada Joint Program Office (AJPO) of the U.S. Department of Defense. This suite contains a large number of tests and concentrates on measuring the execution performance of an Ada system.

AES The Ada Evaluation System is a benchmark suite developed for the United Kingdom's Ministry of Defence. It covers more features of an Ada compilation system than the ACEC, but contains fewer execution performance tests.

The ACEC and AES tests will be merged into a new release of the ACEC test suite.

Appendix 2 on page 207 presents information on the sources for the benchmark suites.

2.1.3 Test Environment

The individual benchmark tests were compiled using the Verdix Ada Development System (VADS), the commercial compiler selected for the BSY-2 test environment and executed on a representative target system. Specifically, a Digital Equipment Corporation clustered microcomputer running the VMS operating system was used as the host system for compilation and data storage. The target system tested a 25 MHz MC68030 microprocessor, and was assembled from commercially available components from Motorola Microsystems.

The MC68030 microprocessor target system was selected to emulate the prototype BSY-2 hardware environment. Cost considerations precluded obtaining military specification hardware for testing. The BSY-2 communications network was not emulated or tested for this study.

The host environment used Digital Equipment Corporation's VMS operating system, one of the two host operating systems used by the BSY-2 development team, but did not specifically emulate any BSY-2 host environment. In general, the host environment was not tested for this report.

The Verdix Ada Development System (VADS) compiler was tested using the host and target systems. The versions of the VADS compiler and run-time system were matched to the version used for BSY-2.

The specifics of the test configuration(s) used are indicated in each performance report.

2.1.4 Accuracy of Data and Results

The benchmark suites used in this report measure and report execution times and, less frequently, memory sizes for their tests. In general, these times are used for drawing all the conclusions in the individual performance reports.

Execution times in each of the benchmark suites were collected by use of the standard Ada clock. This relatively coarse but portable timing source requires that the benchmark measure a sufficiently large

body of code to ensure accurate results. This is achieved by using a large body of representative state-ments or the dual loop paradigm. The specific methods vary among the suites. Thus all times reported in this report represent average (mean) values. The ACEC states that its accuracy is $\pm 5.0\%$ and the PIWG asserts an accuracy of $\pm 10.0\%$. The AES documentation does not set a precise limit, but will mark as "inconsistent" any test where the time for any individual test execution varies by more than 10% from the average time obtained from five executions of a test.

Due to the methods used in obtaining timing data, none of the standard benchmark suites reflects the range of variation at the statement level. Thus the performance data generated by the standard suites are not suitable for worst case analysis. The recommendations in this report therefore reflect choices which should improve average performance.

In several instances, as noted in individual performance reports, errors in the benchmark results were discovered. In other instances, disagreement was noted between the test suites.

2.1.5   Question Format

The individual performance reports use a common format. Instructions for forming and filling out perfor-mance reports are presented in Appendix A on page 181. In reading the reports, the first three sections (Question, Summary, and Discussion) provide a summary view of the entire report along with the most important performance data. The remaining sections describe the test environment and present the rel-evant data from the benchmark suites. These sections may be read selectively, according to the read-er's interests.

2.2 Arrays

Question: What are the performance characteristics of array aggregate assignments and corresponding loop constructs?

Summary: Array aggregate assignment had the best observed performance. For high performance, arrays should be declared locally and accessed directly. Suppressing checks provides moderate performance gains. Packing arrays is particularly effective for Boolean arrays, and generally improves performance and saves space.

Discussion: Arrays are convenient structures for holding large homogenous bodies of data. The assignment, retrieval, and manipulation of array contents, performed many times, affords the programmer the opportunity to improve runtime performance efficiently by concentrating on small, intensively executed sections of code. Similarly, since large arrays consume large amounts of storage, they are natural targets for representation using compact format. Economy of space and time are not independent characteristics, however, and the interaction may require a compromise.

The spectrum of array formats and the operations performed on them is potentially broad, but the following performance characteristics are considered to be of general interest:

- What is the performance of various Ada statements for accessing the elements contained in arrays?

  Examples: Array slices, loop constructs.

  Results: In order of efficiency: use literal assignment (i.e. one statement per array element), array aggregates, array assignments, for loops, while loops. Tests were performed for arrays of integers, reals and Boolean. (Note: however the coverage of the tests was not orthogonal across the data types and used only simple assignments [Observation 5 on page 21]).

- How is runtime performance affected by the form of the range specification?

  Examples: Constants, attributed ranges, ranges held in variables.

  Results: This characteristic was not addressed adequately by the test results available.

- How is runtime performance affected by the method used to create the array?

  Examples: Static allocation in library unit, within declare block, via **new** allocator.

  Results: The AES test TI02 (Observation 2 on page 13) indicates that allocating arrays within the local execution block is more efficient than creating them with a **new** operation or defining them within a library unit.

- What are the performance effects of suppressing checks when manipulating arrays.

  Examples: **pragma** SUPPRESS (INDEX_CHECK), **pragma** SUPPRESS (RANGE_CHECK).

  Results: The ACEC tests perform a number of comparisons between code segments with and without checking enabled (Observation 3 on page 17). While several of the tests proved to be incorrect, the valid tests indicate that the compiler optimizes

---

checks for constant indices, that the overhead for checking while multiplying the contents of a single array element in one statement is 9%, and that the overhead for checking in a statement including references to three elements of the same array is about 24%.

- What are the performance characteristics of packed vs. unpacked arrays?

  Examples: packed vs. unpacked Boolean arrays.

  Results: The AES test TI02 (Observation 2 on page 13) indicates that a packed representation in general does not change or improves runtime performance, with one observed but trivial exception for a mix of array operations. The tested arrays contained Boolean and character values.

  The ACEC tests summarized in Observation 1 indicate that packed Boolean arrays gain significant runtime performance benefits from packing with the exception of accessing individual elements (tests ss333 and ss344). It is not clear however if packing other array types will demonstrate similar *decreases* in execution time.

  The ACEC tests show that packing is effective for arrays of integer (Observation 4 on page 21). Integer types are packed to bytes, but do not span bytes (storage units). Execution speed is not compared between packed and unpacked integer arrays.

For related information, see the entries on loop performance (page 77) and the effects of suppressing constraint checks (page 25).

Configuration(s) tested:

1. **Host**: DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: ACEC tests of operations on Boolean arrays, particularly the effects of packed vs. unpacked arrays.

The ACEC examines execution speed for a number of separate Boolean array operations. These tests show that Boolean arrays generally reduce execution time when packed. The only exception is in accessing individual array elements, where the packed array required about 30% more time to perform the assignment.

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Reports of "Language Feature Overhead":

"Small Boolean Arrays (unpacked vs packed) =, AND, NOT," page 9

"Small Boolean Arrays (unpacked vs packed) =, AND," page 9

The SSA report frequently refers to tests of "16 bit" Boolean arrays. The test code does not force any representation, so the actual array layout used is set by the compiler defaults.

```
------------------------------------------------------------------------

                     Language Feature Overhead

------------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) =, AND, NOT
------------------------------------------------------------------------
Test      Execution    Bar                                     Similar
Name      Time         Chart                                   Groups
------------------------------------------------------------------------
ss337        5.70      ****                                       |
ss326       44.80      ********************************** ***     |
------------------------------------------------------------------------
                     Individual Test Descriptions
------------------------------------------------------------------------

------------------------------------------------------------------
ss337 bool := ( s1 and not s2 )  = s3 ;
      -- operations on 16 bit packed boolean array, =, AND, NOT
------------------------------------------------------------------
ss326 bool := ( b1 and not b2 )  = b3 ;
      -- operations on 16 bit unpacked boolean array, =, AND, NOT
------------------------------------------------------------------


------------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) =, AND
------------------------------------------------------------------------
Test      Execution    Bar                                     Similar
Name      Time         Chart                                   Groups
------------------------------------------------------------------------
ss338        3.61      ******                                     |
ss327       22.20      ********************************** ****     |
------------------------------------------------------------------------
                     Individual Test Descriptions
------------------------------------------------------------------------
```

```
-------------------------------------------------------
ss338 bool := ( s1 and s2 ) = s1 ;
      -- operations on 16 bit packed boolean array, =, AND
-------------------------------------------------------
ss327 bool := ( b1 and b2 ) = b1 ;
      -- operations on 16 bit unpacked boolean array, =, AND
---------------------------------------------------------------
```

```
---------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) /=, AND
---------------------------------------------------------------

Test      Execution   Bar                                    Similar
Name      Time        Chart                                  Groups
---------------------------------------------------------------

ss339        3.38     *****                                     |
ss328       24.50     *********************************         |
---------------------------------------------------------------
                    Individual Test Descriptions
---------------------------------------------------------------
```

```
-------------------------------------------------------
ss339  IF ( s2 AND s3 ) /= s3 THEN die ; END IF ;
       -- operations on 16 bit packed boolean array, /=, AND
-------------------------------------------------------
ss328  IF ( b2 AND b3 ) /= b3 THEN die ; END IF ;
       -- operations on 16 bit unpacked boolean array, /=, AND
-------------------------------------------------------
```

```
---------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) AND
---------------------------------------------------------------

Test      Execution   Bar                                    Similar
Name      Time        Chart                                  Groups
---------------------------------------------------------------

ss340        3.13     ****                                      |
ss329       24.30     *********************************         |
---------------------------------------------------------------
                    Individual Test Descriptions
---------------------------------------------------------------
```

```
-------------------------------------------------------
ss340  s4 := s1 AND s2 ;
       -- operations on 16 bit packed boolean array, AND
-------------------------------------------------------
ss329  b4 := b1 AND b2 ;
       -- operations on 16 bit unpacked boolean array, AND
-------------------------------------------------------
```

```
---------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) OR
---------------------------------------------------------------

Test      Execution   Bar                                    Similar
Name      Time        Chart                                  Groups
---------------------------------------------------------------

ss341        3.13     ****                                      |
ss330       24.30     *********************************         |
---------------------------------------------------------------
                    Individual Test Descriptions
---------------------------------------------------------------
```

```
-------------------------------------------------------
ss341  s4 := s1 OR s2 ;
       -- operations on 16 bit packed boolean array, OR
-------------------------------------------------------
ss330  b4 := b1 OR b2 ;
       -- operations on 16 bit unpacked boolean array, OR
-------------------------------------------------------
```

```
-------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) OR
-------------------------------------------------------------------
Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
-------------------------------------------------------------------
ss342        52.70       *********×***********************    |
ss331        54.50       *********************************    |
-------------------------------------------------------------------
                    Individual Test Descriptions
-------------------------------------------------------------------
-- OR (aggregate with range clause)
bool := yy > zz ;
-------------------------------------------------------------------
ss331   b4 := b1 OR set1'( set1'range => bool ) ;
        -- operations on 16 bit unpacked boolean array, OR
        -- uses an aggregate with range clause
-------------------------------------------------------------------
ss342   s4 := s1 OR set2'( set2'range => bool ) ;
        -- operations on 16 bit packed boolean array, OR
        -- uses an aggregate with range clause
-------------------------------------------------------------------


-------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) XOR
-------------------------------------------------------------------
Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
-------------------------------------------------------------------
ss343         3.33       ****                                 |
ss332        26.50       **********************************   |
-------------------------------------------------------------------
                    Individual Test Descriptions
-------------------------------------------------------------------
-------------------------------------------------------------------
ss343   s4 := s1 XOR s2 ;
        -- operations on 16 bit packed boolean array, XOR
-------------------------------------------------------------------
ss332   b4 := b1 XOR b2 ;
        -- operations on 16 bit unpacked boolean array, XOR
-------------------------------------------------------------------


-------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) Fetch and Store
-------------------------------------------------------------------
Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
-------------------------------------------------------------------
ss333         1.69       **********                           |
ss344         5.78       **********************************   |
-------------------------------------------------------------------
                    Individual Test Descriptions
-------------------------------------------------------------------
-------------------------------------------------------------------
ss344   s1 ( ei ) := s1 ( ej ) ;
        -- fetch from and store into indexed element (16 bit packed)
-------------------------------------------------------------------
ss333   b1 ( ei ) := b1 ( ej ) ;
        -- fetch from and store into indexed element (16 bit unpacked)
-------------------------------------------------------------------


-------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) Slice Assignment
-------------------------------------------------------------------
Test        Execution    Bar                                  Similar
```

```
Name       Time       Chart                                          Groups
-----------------------------------------------------------------------------
ss345      2.31       ********************                             |
ss334      3.96       **********************************              |
-----------------------------------------------------------------------------
                           Individual Test Descriptions
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
ss345   s1 ( 10..14 ) := s2 ( 11..15 ) ;
           -- operations on 16 bit packed boolean array, slice assignment
-----------------------------------------------------------------------------
ss334   b1 ( 10..14 ) := b2 ( 11..15 ) ;
           -- operations on 16 bit unpacked boolean array, slice assignment
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) - Conversion
-----------------------------------------------------------------------------
Test       Execution  Bar                                            Similar
Name       Time       Chart                                          Groups
-----------------------------------------------------------------------------
ss335      50.60      ***********************                         |
ss346      76.00      ***********************************             |
-----------------------------------------------------------------------------
                           Individual Test Descriptions
-----------------------------------------------------------------------------
Test       Execution  Bar                                            Similar
Name       Time       Chart                                          Groups
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
ss335   b4 := set1 ( s1 ) ;
           -- convert from unpacked to packed 16 bit boolean array
-----------------------------------------------------------------------------
ss346   s4 := set2 ( b1 ) ;
           -- conversion packed to unpacked 16 bit boolean array
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
Small Boolean Arrays (unpacked vs packed) - Fetch From Array
-----------------------------------------------------------------------------
Test       Execution  Bar                                            Similar
Name       Time       Chart                                          Groups
-----------------------------------------------------------------------------
ss336      1.28       ***************************                      |
ss347      1.67       **********************************               |
-----------------------------------------------------------------------------
                           Individual Test Descriptions
-----------------------------------------------------------------------------
ss347   bool := s5 ( ei ) ;
           -- fetch element from 16 bit packed boolean array
-----------------------------------------------------------------------------
ss336   bool := b5 ( ei ) ;
           -- fetch element from 16 bit unpacked boolean array
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
Large Boolean Arrays (unpacked vs packed) AND
-----------------------------------------------------------------------------
Test       Execution  Bar                                            Similar
Name       Time       Chart                                          Groups
-----------------------------------------------------------------------------
ss351      58.10      **********                                       |
ss348      202.40     **********************************              |
-----------------------------------------------------------------------------
```

```
                        Individual Test Descriptions
-----------------------------------------------------------------------------
-------------- ---------------------------------------------
ss348 bool := ( lb1 AND lb2 ) = lb1 ;
          -- operations on large unpacked Boolean array, =, AND
-----------------------------------------------------------------
ss351 b. _: := ( ls1 AND ls2 ) = ls1 ;
          -- operations on large packed Boolean array, =, AND
-----------------------------------------------------------------------------


--------------------------------------------------------------------------
Large Boolean Arrays (unpacked vs packed) AND, NOT, OR, XOR
-----------------------------------------------------------------------------

Test      Execution     Bar                                        Similar
Name      Time          Chart                                      Groups
-----------------------------------------------------------------------------
ss352       406.40      *****************                       |
ss349       789.40      ************************************     |
-----------------------------------------------------------------------------
                        Individual Test Descriptions
-----------------------------------------------------------------------------
-----------------------------------------------------------------
ss352    ls4 := ( NOT ( ls1 AND ls2 ) OR ls3 ) XOR ls5 ;
          -- operations on large packed Boolean array, AND, OR, XOR
-----------------------------------------------------------------
ss349    lb4 := ( NOT ( lb1 AND lb2 ) OR lb3 ) XOR lb5 ;
          -- operations on large unpacked Boolean array, NOT, XOR, AND, OR
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
Large Boolean Arrays (unpacked vs packed) AND
-----------------------------------------------------------------------------

Test      Execution     Bar                                        Similar
Name      Time          Chart                                      Groups
-----------------------------------------------------------------------------
ss350         2.18      ****************************            |
ss353         2.57      ********************************        |
-----------------------------------------------------------------------------
                        Individual Test Descriptions
-----------------------------------------------------------------------------
-------------------------------------------------------------
ss350    lb4 ( ei ) := ls1 ( ej ) ;
          -- convert packed to unpacked large Boolean array
-------------------------------------------------------------
ss353    ls4 ( ei ) := lb1 ( ej ) ;
          -- convert large unpacked Boolean array to packed
-----------------------------------------------------------------------------
```

Observation 2: AES tests for array performance using a combined workload.

In summary, the test results indicate that the arrays allocated within the execution scope and directly accessed via indexing (rather than via access variables) generally show the best run-time performance. A packed array may be manipulated as quickly or more quickly than an unpacked array for all tested cases (the exception is when access is via array slices for arrays allocated "on stack"). This effect is especially pronounced for Boolean arrays.

The AES uses a consistent format for its tests of array operations. Instead of testing a single operation (e.g., assignment of an integer to a one-dimensional array), a mix of many related operations is performed (e.g., assignments to one-, two-, and three-dimensional arrays). These test sequences are ap-

plied to arrays that are allocated differently along two dimensions: representation (packed vs. unpacked arrays), and creation (three types of declaration). (The AES output refers to these declarations as "On-Stack," "Library Record," and "Heap Record" [see below].) The timings provided as output from these tests are useful for providing information on the runtime performance trade-offs between allocation strategies; however, the measured times represent a mix of instructions that cannot be related to individual array operations. The times are also usefully employed when the test is performed on different releases of software and/or hardware and the results compared.

The AES uses three allocation methods for test arrays (the following examples are for unpacked arrays; the packed forms use the same form with the addition of **pragma** PACK).

Common type declarations:

```
type ONE_DIM_U is array (1 .. 5) of CHARACTER;
type TWO_DIM_U is array (0 .. 3) of ONE_DIM_U;
type THREE_DIM_U is array (BOOLEAN) of TWO_DIM_U;

-- The access types required for this test

type REF_ONE_DIM_U is access ONE_DIM_U;
type REF_TWO_DIM_U is access TWO_DIM_U;
type REF_THREE_DIM_U is access THREE_DIM_U;
```

- **AES "On Stack" Allocation**

```
declare

    -- Variables of the required array types declared on the stack
    ONE_U_1, ONE_U_2 : ONE_DIM_U;
    TWO_U_1, TWO_U_2 : TWO_DIM_U;
    THREE_U_1        : THREE_DIM_U;

begin
    -- timed test;
end;
```

- **AES "Library Array" Allocation**

```
declare

    -- Variables of the required array types are
    -- declared in the following library.

    use TI02AA; -- library unit containing all array
                -- and pointer definitions and declarations

begin
    -- timed test;
end;
```

- **AES "Heap Record" Allocation**

```
declare

    -- Variables of the required array types declared on the heap
    ONE_U_1, ONE_U_2 : REF_ONE_DIM_U;
    TWO_U_1, TWO_U_2 : REF_TWO_DIM_U;
    THREE_U_1        : REF_THREE_DIM_U;
```

```
begin
    -- timed test;
end;
```

Declarations in procedures were not tested:

```
procedure TI02A is

    ONE_U_1, ONE_U_2 : ONE_DIM_U;
    TWO_U_1, TWO_U_2 : TWO_DIM_U;
    THREE_U_1        : THREE_DIM_U;

begin
    -- timed test;
end;
```

The reason for this omission is not known.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Tests TI02A-F.

I.5. TI02A

This test examines the efficiency of array object
manipulation, in particular, the cpu time taken in
component indexing. Timings are taken for both packed and
unpacked arrays, declared on the stack, on the heap and
in library packages. One-, two- and three-dimensional
character arrays are used.

Array Component Indexing:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	664us	961us	977us
Packed	664us	890us	665us

I.6. TI02B

This test examines the efficiency of array object
manipulation, in particular, the cpu time taken in array
assignments. Timings are taken for both packed and
unpacked arrays, declared on the stack, on the heap and
in library packages. One-, two- and three-dimensional
character arrays are used.

Array Assignment:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	168us	170us	182us
Packed	168us	170us	183us

I.7. TI02C

This test examines the efficiency of array object

manipulation, in particular, the cpu time taken in array comparisons. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages. One-, two- and three-dimensional character arrays are used.

Character Array Comparison:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	410ns	410ns	410ns
Packed	410ns	410ns	410ns

I.8. TI02D

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in performing logical operations on boolean arrays. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages.

Logical Operations on Boolean Arrays:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	587us	628us	805us
Packed	196us	203us	231us

I.9. TI02E

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in array concatenation. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages. An unconstrained one-dimensional integer array is used.

Test failed. Wrong TEST_ID in TEST.TST

I.10. TI02F

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in array slicing. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages. An unconstrained one-dimensional integer array is used.

Array Slicing:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	566us	573us	1.48ms
Packed	584us	573us	1.48ms

Observation 3: ACEC tests comparing operations on arrays with and without constraint checking.

These ACEC tests provide a comparison between instruction sequences with checks enabled and checks disabled. However, due to errors in the command files provided for running the tests, the majority of the comparisons are invalid (see below). Useful data was obtained from SSA report "3 References To Same Array In Expression," SSA report "Arrays - Assignment To 1 Dimensional Array of Real" for test ss55 compared with ss169 and SSA report "1D Array (Same Index) Both Sides of Assignment Stmt."

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Reports of "Language Feature Overhead":

> "Array of BigInt," page 17
>
> "Array of Character Strings," page 18
>
> "3 References to Same Array in Expression," page 18
>
> "Arrays - Assignment to 1 Dimensional Array of Real," page 19
>
> "Arrays - 2 Dimensional Arrays of Real," page 19
>
> "Arrays - Assignment to Array of Real," page 20
>
> "1D Array (Same Index) Both Sides of Assignment Stmt," page 20.

Note that several tests are invalid since checks were disabled for all of the tests performed. This was due to an error in the control files provided with the test suite. These SSA reports are marked with the phrase ☞**Invalid result, checks disabled**.

```
---------   -----------------------------------------------------------------

                        Language Feature Overhead

---------------------------------------------------------------------------
```
Array Of bigInt ☞**Invalid result, checks disabled**
```
---------------------------------------------------------------------------
Test        Execution    Bar                                    Similar
Name        Time         Chart                                  Groups
---------------------------------------------------------------------------
ss53          1.28       ******************************           !
ss284         1.28       ******************************           !
ss54          1.42       ********************************         !
ss285         1.42       ********************************         !
---------------------------------------------------------------------------
```

```
                    Individual Test Descriptions
--------------------------------------------------------------------
--------------------------------------------------------------
ss53 ii := il ( ei ) ;
   -- Reference to subscripted array of int, without checking.
------------------------------------------------------------------
ss54 ii := il ( ei + 1 ) :
   -- Reference to subscripted array of int, without checking.
------------------------------------------------------------------
ss284 li := lil ( bigint ( ei ) ) ;
   -- fetch from array of bigint.
------------------------------------------------------------------
ss285 li := lil ( bigint ( ei + 1 ) ) ;
   -- fetch from array of bigint, fold term into address computation
--------------------------------------------------------------------


------------------------------------------------------------------
Array of character strings ☞Invalid result, checks disabled
--------------------------------------------------------------------

Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
--------------------------------------------------------------------

ss243         1.28       **********************************  |
ss53          1.28       **********************************  |
--------------------------------------------------------------------
                    Individual Test Descriptions
--------------------------------------------------------------------
Compare reference to array of integers to an array of a subtype of string.

SUBTYPE c2 IS String ( 1 .. 2 ) ;
TYPE trans_type IS ARRAY ( int'(0) .. int'(255) ) OF c2 ;
trans : trans_type ;
TYPE rec_array IS ARRAY ( int'(1)..int'(4) ) OF byte ;
SUBTYPE c8 IS String ( 1 .. 8 ) ;
ccc, hex : c8 ;
------------------------------------------------------------------
ss53  ii := il ( ei ) ;
   -- Reference to subscripted array of int, without checking.
------------------------------------------------------------------
ss243 hex ( 1 .. 2 ) := trans ( ll ) ;
   -- access to array of 2 character strings and assign to a slice
--------------------------------------------------------------------


------------------------------------------------------------------
3 References To Same Array In Expression
--------------------------------------------------------------------

Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
--------------------------------------------------------------------

ss193         8.52       ****************************        |
ss174        10.60       ***********************************  |
--------------------------------------------------------------------
                    Individual Test Descriptions
--------------------------------------------------------------------
ss174 xx := el ( ei + 2 ) + el ( ei + 1 ) + el ( ei ) ;
   -- 3 references to same array in expression, subscripting
   -- expression has constant terms with subscript range
   -- checking enabled.  Bounds checks can be merged.
------------------------------------------------------------------
ss193 xx := el ( ei + 2 ) + el ( ei + 1 ) + el ( ei ) ;
   -- 3 references to same array in expression, subscripting
   -- expression has constant terms with subscript range checking
   -- suppressed.  Subscripting expression has common subexpression.
------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Arrays - Assignment To 1 Dimensional Array Of Real
----------------------------------------------------------------------
Test      Execution    Bar                                  Similar
Name      Time         Chart                                Groups
----------------------------------------------------------------------
ss55        0.78      ****                                     |
ss169       0.78      ****                                     |
ss645       1.28      *******                                  |
ss53        1.28      *******                                  |
ss309       1.28      *******                                  |
ss54        1.42      ********                                  |
ss646       3.95      ********************                       |
ss647       6.26      **********************************          |
----------------------------------------------------------------------
                     Individual Test Descriptions
----------------------------------------------------------------------

--------------------------------------------------------------
ss53   ii := i1 ( ei ) ;
    -- Reference to subscripted array of int, without checking.
--------------------------------------------------------------
ss54   ii := i1 ( ei + 1 ) ;
    -- Reference to subscripted array of int, without checking.
--------------------------------------------------------------
ss55   ii := i1 ( 1 ) ;
    -- Reference array with a constant subscript, without checking.
--------------------------------------------------------------
ss169  ii := i1 ( 1 ) ;
    -- fetch from 1D array with range checking, using constant subscript
--------------------------------------------------------------
ss173 ii := i1 ( ei + 1 ) ;
    -- Reference to subscripted array of int, with checking.
--------------------------------------------------------------
ss309  hue := stat ( ei ) ;
    -- access array of an enumerated type
--------------------------------------------------------------
ss645  one := e1 ( ei ) ;
    -- fetch from 1D array, checking suppressed
--------------------------------------------------------------
ss646  one := e2 ( ei , ej ) ;
    -- fetch from 2D array, checking suppressed
--------------------------------------------------------------
ss647  one := e3 ( ei , ej , ek ) ;
    -- fetch from 3D array, checking suppressed
----------------------------------------------------------------------


----------------------------------------------------------------------
Arrays - 2 Dimensional Arrays Of Real ☞Invalid result, checks disabled
----------------------------------------------------------------------
Test      Execution    Bar                                  Similar
Name      Time         Chart                                Groups
----------------------------------------------------------------------
ss646       3.95      *********************************   |
ss759       3.96      *********************************    |
ss762       3.96      *********************************    |
----------------------------------------------------------------------
                     Individual Test Descriptions
----------------------------------------------------------------------
e2 : ARRAY ( int'(1)..int'(10) ,int'(1)..int'(10) ) OF real
           := ( int'(1)..int'(10) =>( int'(1)..int'(10) =>1.0));
ei, ej, ek : int := 1;
--------------------------------------------------------------
ss646 one := e2 ( ei, ej ) ;
```

```
        -- fetch from 2D array : No constraint checking.
        -- ss646 and ss759 SHOULD take the same time.
        --------------------------------------------------------
ss759 one := e2 ( ei, ej ) ;
        -- Fetch value from two dimensional floating point array.
        -- No constraint checking.
        --------------------------------------------------------
ss762 one := e2 ( ei, ej ) ;
        -- Fetch value from two dimensional floating point array.
        -- Constraint checking.
        -----------------------------------------------------------------
```

```
        -----------------------------------------------------------------
```

Arrays - Assignment to Array of Real ☞**Invalid result, checks disabled**

```
-------------------------------------------------------------------
```

Test Name	Execution Time	Bar Chart	Similar Groups
ss761	1.28	*******	¦
ss758	1.29	*******	¦
ss759	3.96	**********************	¦
ss762	3.96	**********************	¦
ss760	6.27	**************************************	¦
ss763	6.28	**************************************	¦

```
---------------------------------------------------------------
```
                        Individual Test Descriptions
```
---------------------------------------------------------------
---------------------------------------------------------------
```

```
ss758 one := el ( ei ) ;              -- No constraint checking.
        -- Fetch from one dimensional array.
        -------------------------------------------------------
ss759 one := e2 ( ei , ej ) ;         -- No constraint checking.
        -- Fetch from two dimensional floating point array.
        -------------------------------------------------------
ss760 one := e3 ( ei , ej , ek ) ;    -- No constraint checking.
        -- Fetch from three dimensional floating point array.
        -------------------------------------------------------
ss761 one := el ( ei ) ;              -- Constraint checking.
        -- Fetch from one dimensional array.
        -------------------------------------------------------
ss762 one := e2 ( ei , ej ) ;         -- Constraint checking.
        -- Fetch from two dimensional floating point array.
        -------------------------------------------------------
ss763 one := e3 ( ei , ej , ek ) ;    -- Constraint checking.
        -- Fetch from three dimensional floating point array.
        -------------------------------------------------------
```

```
---------------------------------------------------------------
```
1D Array (Same Index) Both Sides Of Assignment Stmt
```
-------------------------------------------------------------------
```

Test Name	Execution Time	Bar Chart	Similar Groups
ss192	7.21	******************************* ¹***	¦
ss170	7.89	********************************* *******	¦

```
---------------------------------------------------------------
```
                        Individual Test Descriptions
```
---------------------------------------------------------------
---------------------------------------------------------------
```

```
ss170 el ( ei ) := el ( ei ) * one ;
        -- Fetch from and store into 1D array (same index) on both
        -- left and right side of assignment statement with
        -- subscript range checking enabled. Subscript
        -- computation need only be verified once.
```

```
----------------------------------------------------------------
ss192 el ( ei ) := el ( ei ) * one ;
     -- Same subscripting expression of left and right side of
     -- assignment statement.  Checking suppressed.
----------------------------------------------------------------
```

Observation 4: ACEC tests of the effectiveness of packed representation.

The tests included here all specify a small array of unsigned integer values, where the base type is a range of $0..2^n$. The values of n tested are 3, 5, 7, and 16, but not all test results were available for the target configuration.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Ancillary Data."

Identifiers of the form "a_#" indicate the base type for packing, where # is the minimum number of bits required to hold the value. Identifiers of the form "a#(1)" indicate a cell of an array of the base type.

> Note that the percentage value in "packing achieves x% of maximum" for the SSA report Ancillary Data is computed incorrectly and should be ignored.

```
----------------------------------------------------------------

                         Ancillary Data

----------------------------------------------------------------
Ancillary Data - List
----------------------------------------------------------------


----------------------------------------------------------------
ss657  a_5'size  = 8,   a5(1)'size  = 8, packing achieves100% of maximum
----------------------------------------------------------------
ss662  a_7'size  = 8,   a7(1)'size  = 8, packing achieves100% of maximum
----------------------------------------------------------------
ss672  a_15'size =16,   a15(1)'size =16, packing achieves100% of maximum
----------------------------------------------------------------
ss677  a_16'size =16,   a16(1)'size =16, packing achieves100% of maximum
----------------------------------------------------------------
```

Observation 5: ACEC tests of array access methods.

These tests use simple assignments and various types of shifts to iterate through the array. While some variation is checked, including arrays of real, integer, and Boolean quantities and checks on and off, the coverage is not orthogonal. Also, note that the SSA report "Array Assignment" reports results from assignments to two types of arrays: e1, which holds floating point numbers, and i1, which holds integers.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Reports of "Coding Style Variations":

"Array Assignment"

"Shift Packed Boolean Array"

```
----------------------------------------------------------------------
                       Coding Style Variations

----------------------------------------------------------------------
Array Assignment
----------------------------------------------------------------------
Test     Execution   Bar                                      Similar
Name     Time        Chart                                    Groups
----------------------------------------------------------------------
ss81        7.56     **********                           |
ss171       7.66     **********                             |
ss388       7.80     **********                               |
ss77        8.83     ************                                |
ss78        8.83     ************                                |
ss79        8.83     ************                                |
ss80        9.93     *************                                  |
ss209      25.50     *********************************            |
----------------------------------------------------------------------
                     Individual Test Descriptions
----------------------------------------------------------------------
All of these represent different ways of assigning an array of ten
elements to one. ss81, ss171, and ss209 refer to integers, the rest
to reals.
----------------------------------------------------------------------
ss77   el := ( 1..10 => 1.0 ) ;
       -- aggregate with range specification
----------------------------------------------------------------------
ss78   el := ( 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 ) ;
       -- aggregate with elements positionally specified
----------------------------------------------------------------------
ss79   el := xel ;
       -- copy array
----------------------------------------------------------------------
ss80   FOR i IN 1..10 LOOP    el ( i ) := 1.0 ;    END LOOP;
       -- Array assignment using a FOR loop to set each element to 1.0.
----------------------------------------------------------------------
ss81   FOR i IN 1..10 LOOP    il ( i ) := i ;    END LOOP ;
       -- Array assignment using a FOR loop to set the "ith" element to "i".
----------------------------------------------------------------------
ss171  FOR i IN 1..10 LOOP    il ( i ) := i ;  END LOOP ;
       -- subscript with FOR loop index (in range) compile time range
            check possible
----------------------------------------------------------------------
ss209  ii := 1 ;
          WHILE ii <= 10
          LOOP
             il ( ii ) :=ii ;
             ii := ii + 1 ;
          END LOOP ;
       -- WHILE loop comparable to the FOR loop in ss81
----------------------------------------------------------------------
ss388  el ( 1 ) := 1.0 ;  el ( 2 ) := 1.0 ; ... el ( 10 ) := 1.0 ;
       -- sequence of literal assignment statements to array components.
----------------------------------------------------------------------


----------------------------------------------------------------------
   Shift Packed Boolean Array
----------------------------------------------------------------------
Test     Execution   Bar                                      Similar
```

```
Name       Time      Chart                                        Groups
---------------------------------------------------------------------------
ss524       3.38     *                                              |
ss525      83.00     ***********************************            |
---------------------------------------------------------------------------
                      Individual Test Descriptions
---------------------------------------------------------------------------
Two alternative ways of performing the same operation, one using
slice aggregate assignments, the other using an element by element
loop.

TYPE ba16_type IS ARRAY ( 1..16 ) OF Boolean ;
PRAGMA pack ( ba16_type ) ;
ba : ba16_type ;
------------------------------------------------------------
ss524   ba ( 1..15 ) := ba ( 2..16 ) ;   ba ( 16 ) := False ;
   -- Shift a packed Boolean array using slice assignments.
   -- Could be implemented as integer divide.
------------------------------------------------------------
ss525
   FOR i IN 1..15            264.1
   LOOP
      ba ( i ) := ba ( i + 1 ) ;
   END LOOP ;
   ba ( 16 ) := False ;
   -- Shift a packed Boolean array using a FOR loop and
   -- element by element assignment.
---------------------------------------------------------------------------
```

References

- none

2.3 Check Suppression

Question: How does performance change when checks are turned off?

Summary: Suppressing runtime checks can increase execution speed. However, the increase is dependent on the interaction of compiler optimization and the operations and data subject to runtime checking. Removal of constraint checks also generally decreases the size of the executable code.

Discussion: Constraint checking is often cited as reducing program performance. Before suppressing checks, it is useful to know the actual cost of constraint checking. The AES test used in this report times code segments which use many individual checks. They may be considered representative of a worst case use of checks. Similarly, the information on code size changes gives an indication of the amount of space checking code requires.

There are several anomalous results. In one case this is an error in the AES test TO07. The others represent unexpected behavior on the part of the Verdix Ada Development System.

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: AES Test TO07, found that for representative blocks of code the suppression of all checks and selected checks improves performance in most cases.

As can be seen from the AES output below, test execution time improved when INDEX_CHECK, ACCESS_CHECK, DISCRIMINANT_CHECK, LENGTH_CHECK, and RANGE_CHECK were individu ′ or collectively suppressed. However, execution times did not improve when OVERFLOW_CHECK and STORAGE_CHECK were suppressed. The measured time unexpectedly increased when DIVISION_CHECK was individually suppressed. Observation 2 examines these performance anomalies.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0, Test TO07 ~ (9 digits of precision).

```
0.7.  TO07

      This test determines whether pragma SUPPRESS makes any
      appreciable difference to the efficiency of Ada code.

      In  addition  to  pragma  SUPPRESS,  the  following
      implementation  dependent  pragma  was  used  during  the
      execution of this test.
```

```
pragma SUPPRESS(ALL_CHECKS);
```

The following table shows the CPU times to execute blocks
of code when the specified checks are left in and when
they are suppressed.

It also shows the time for each block of code when the
statement pragma SUPPRESS(ALL_CHECKS); is included.
In each case the block of code timed makes heavy use of
statements that require the specified type of checking.

Type of check	With the check in	With check suppressed	All checks suppressed
INDEX_CHECK	1.97s	1.03s	1.63s
ACCESS_CHECK	1.08s	961ms	961ms
DISCRIMINANT_CHECK	1.69s	1.27s	1.27s
LENGTH_CHECK	1.82s	1.71s	1.29s
*** RANGE_CHECK	1.64s	145ms	148ms
DIVISION_CHECK	1.44s	1.57s	1.36s
OVERFLOW_CHECK	1.55s	1.55s	1.55s
STORAGE_CHECK	1.49s	1.49s	1.49s

*** This test is erroneous. See Observation 2.

Observation 2: The executable code from AES test TO07 was examined with the VADS debugger for size and performance effects. Suppressing checks causes code size to change, ranging from a decrease of 56% to an *increase* of 14%. Both individual check suppression and full suppression were tested. Further examination of the code explains some of the performance anomalies noted in Observation 1.

The AES test harness performs two separate runs of test TO07. The first examines when checks are individually suppressed, the second run suppresses all checks and repeats the individual timings. The code segment timed for each class of checks is intended to exercise the individual check. The INDEX_-CHECK section performs assignments to individual array elements, for example.

The test avoids actually raising any exception.

As defined in the Ada Language Reference Manual, **pragma** SUPPRESS disables checking only within the innermost enclosing declarative region. Therefore other checks should be active during individual tests, if applicable to the tested operation. This rule holds true in the examined code.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Test TI07.

Table 1: Generated Code Sizes AES Test TO07

Check	Checking On	Individual Check Off		All Checks Off	
	# Bytes	# Bytes	% On[a]	# Bytes	% On[b]
INDEX_CHECK	1760	770	43.8%	1120	63.6%
ACCESS_CHECK	170	122	71.8%	122	71.8%
DISCRIMINANT_CHECK	210	120	57.1%	120	57.1%
LENGTH_CHECK	4032	3648	90.5%	2440	60.5%
RANGE_CHECK	352	0	0.0%[c]	0	0.0%[c]
DIVISION_CHECK	1036	1180	113.9%	816	78.8%
OVERFLOW_CHECK	456	456	100.0%	430	94.3%
STORAGE_CHECK[d]	112	112	100.0%	100	89.3%

a % On = (# bytes checking on / # bytes individual check off) * 100
b % On = (# bytes checking on / # bytes all checks off) * 100
c Invalid test result.
d Size does not include system routines called to perform runtime allocations.

The code sizes are the number of bytes for the test code only, excluding looping and anti-optimization code. It should be noted that the looping and anti-optimization code is located in a region where checks are suppressed and can potentially benefit from suppressing checks, but should not be affected by testing the suppression most individual checks (with the possible exception of INDEX_CHECK). For allocation of new objects (used to exercise STORAGE_CHECK), a call to a run time allocation routine was made. The size of this routine was not known and is not included in the recorded size.

Explanations for a number of individual anomalies were sought:

- Why does suppressing INDEX_CHECK individually yield a better execution time than suppressing all checks?

  Less code is generated by the compiler when the check is individually suppressed than when all checks are suppressed.

- Why are execution times greatly reduced when RANGE_CHECK is suppressed?

The test was optimized to an empty loop when checking was suppressed, invalidating the test. The test performs assignment to locally declared variables, which are not referenced after assignment. With checking on, an exception could occur, so the loop could not be removed. Without range checking, and with the assignment value held constant, the compiler eliminated the assignment.

- Why does execution time increase when DIVISION_CHECK is individually suppressed?

    The code size expanded, resulting in a larger test. The reason for increased code size is not known.

- Why does the execution time remain unchanged when OVERFLOW_CHECK is suppressed (both individually and all checks)?

    The size of code generated is the same when the individual check is suppressed and is only 5.3% less when all checks are suppressed. The 68030 instruction set includes a single instruction which can test the overflow condition, resulting in the small expansion factor. The AES test was not sensitive enough to detect the small difference in code size.

- Why does the execution time remain unchanged when STORAGE_CHECK is suppressed (both individually and when all checks are suppressed)?

    This test uses the new allocator to test storage check. For VADS, this involves a call to a run time routine for heap allocation. The code for this routine was not available for inspection. However the expansion of the visible code showed no increase in size when the STORAGE_CHECK was suppressed and only a 10.7% reduction in size for suppressing all checks. Assuming a relatively constant value for the execution time of the allocation routine, the AES does not detect the small difference in code size.

Observation 3: Static instruction timing was examined for selected code segments of AES test TO07 using configuration #1 (DIY_AES Version 2.0). The calculation was developed by examining test code with the VADS debugger, then deriving the static timing according to the method described in Chapter 11 of the *MC68030 Enhanced 32-Bit Microprocessor User's Manual, Second Edition.* Chapter 11 provides a set of rules for timing which allows the user to account for the pipelined execution of the microprocessor by combining times of adjacent instructions which are eligible for overlapping execution. Additional work must be performed to account for addressing modes used and instructions with data dependent execution time.

Manual calculation of times was unsatisfactory for a number of reasons:

1. The calculation was time consuming and could not be verified. (It was not possible to analyze any of the individual test sequences completely.)

2. The method provided uses time values that do not account for the alignment of instructions on word boundaries in memory.

3. Assumptions must be made about the effects of the instruction and data caches, and are not empirically verifiable.

Automated support for time computation may make static timing analysis practical.

Simple observation of code expansion factors, derived through the VADS debugger, proved to be sufficient for examining timing anomalies.

References

[ANSI]   ANSI; *The American National Standard for the Ada Programming Language*; American National Standards Institute Inc.; 1430 Broadway, New York 10018; 1983.

[Motorola]   Motorola; *MC68030 Enhanced 32-Bit Microprocessor User's Manual, Second Edition*; Prentice Hall, Englewood Cliffs, NJ 07632; 1989.

2.4 Data Location

Question: What is the performance effect of declaring data locally or outside the executing scope?

Summary: Simple data objects are accessed most efficiently when declared in a local block. Operations performed on variables declared in library packages are slightly less efficient. Local declarations, using the **new** operator, are the least efficiently accessed, and are subject to possible storage exhaustion in long running applications.

Discussion: In Ada, programs may declare variables and constants in several locations within the program unit:

- In a package specification or in the declarative part of a package body
- In the declarative part of a subprogram or task body
- In the declarative part of a block
- Implicitly in loop statements

Although the syntax of declarations is the same for each case (with the exception of implicit declaration in loop statements), compilers may use different strategies for data declarations depending upon their location. For instance, declarations in library units may be treated differently than those local to a block.

Performance might also vary according to the type of the object declared. Ada allows for the declaration of simple variables, arrays and records. Objects may be referenced indirectly via pointers, and the user can specify the representation of variables.

In order to reduce this question to a tractable size, emphasis was placed on examining a limited number of data types (simple variables, records, and arrays) declared in a small number of locations (within a local block, at the start of a procedure, and in a library unit). This follows the model provided by the Ada Evaluation System.

The AES tests T!01A-D and TI02A-F examine the performance of records and arrays allocated in different locations (see Observation 1 on page 34). Each test compares the three types of storage and tests two representations of the object (e.g., packed vs. unpacked for arrays). This makes a total of two triplets of matched observations for each test, with six sub-tests for each test.

Figure 2-1 on page 32 summarizes the results from AES tests TI01 and TI02. Each type of storage is ranked within the test (from fastest to slowest) and a summary bar chart presents the counts for each storage type. Observation 1 on page 34 contains the data used to prepare the chart and also presents the Ada declarations used by the AES for "stack" (declaration in a block); "library" (**use** statement within a block) and "heap" (**new** allocator within a block).

Figure 2-1: Summary Graph of AES Tests TI01 and TI02
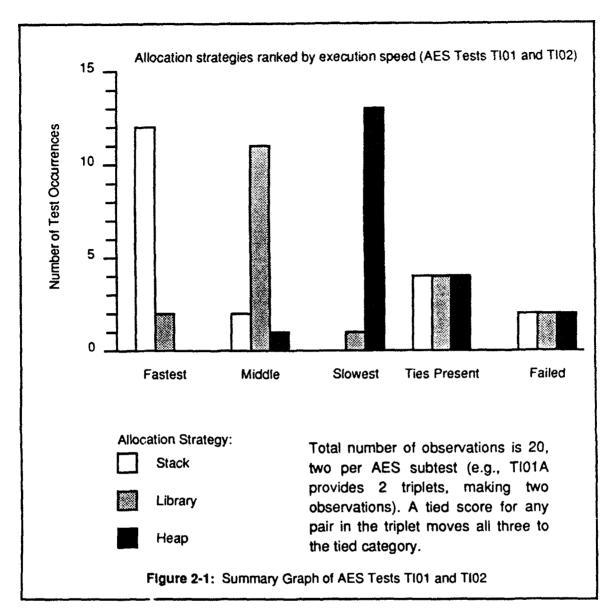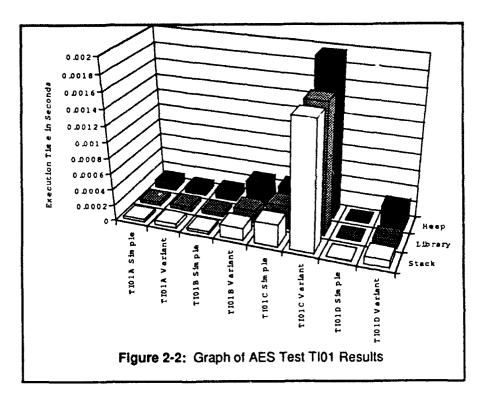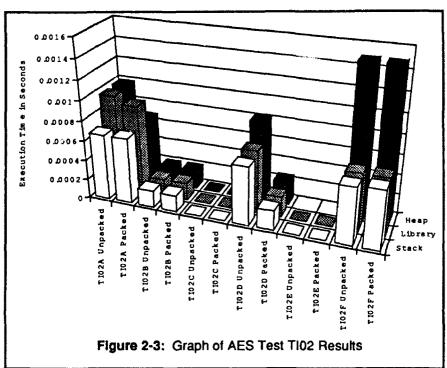
Figure 2 and Figure 2-3 on page 33 show the individual test times for AES tests TI01 and TI02 summarized in Figure 2-1. For comparison purposes, it should be noted that "heap" allocations were created by use of access types, while "stack" and "library" allocations were created by using regular declarations. Any conclusion about heap allocation using the AES data is necessarily also an observation comparing standard type records and arrays with variables addressed by access (pointer) types.

Figure 2-2: Graph of AES Test TI01 Results

Figure 2-3: Graph of AES Test TI02 Results

The AES tests TM05 and TM07 (Observation 2 on page 38) check for potentially undesirable heap storage characteristics. The tests indicate that heap storage creep (storage not returned to the free list after

deallocation) and fragmentation (division of available free storage into unusable small fragments) can occur.

The ACEC SSA Report entry titled "Reference Variables in Different Packages" (Observation 3 on page 39) compares simple variables and small arrays declared locally and in other packages. It is difficult to determine how to group comparable tests. In the single clearly comparable case, tests ss469 and ss470 show no difference in performance between simple variables declared locally and in an outside package.

The ACEC SSA Report entry titled "Reference 0th .. 1024 Real Variable in Package" (Observation 4 on page 40) indicates that the order of declarations in a package (the AES library declaration) has no effect on performance.

Configuration(s) tested:

1.   **Host**: DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

    Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

    Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: AES tests of performance for several types of variable declarations.

The AES tests three locations for declaring variables:

- **Stack** -- The AES uses a declaration of the form:

```
type ARRAY_TYPE is ...;

declare
  AN_ARRAY: ARRAY_TYPE;
begin
  -- Test of performance using AN_ARRAY
end;
```

- **Library** -- The AES uses a declaration of the form:

```
package LIBRARY_ARRAY is
  type ARRAY_TYPE is ...;
  LIB_ARRAY: ARRAY_TYPE;
end LIBRARY_ARRAY;

with LIBRARY_ARRAY;
declare
  use LIBRARY_ARRAY;
begin
  -- Test of performance using LIB_ARRAY
end;
```

- **Heap** -- The AES uses a declaration of the form:

```
type ARRAY_TYPE is ...;
type ARRAY_POINTER is access ARRAY_TYPE;

declare
  REF_ARRAY: ARRAY_POINTER;
begin
  REF_ARRAY := new ARRAY_TYPE;
  -- Test of performance using REF_ARRAY
end;
```

Note that "heap" allocation uses access types and the "stack" and "library" allocation use direct declaration. Thus comparison of heap with stack and library allocation is also a comparison of access types and "regular" variables.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Group I, Tests TI01-2.

```
I. Group I - runtime Efficiency Tests

I.1.  TI01A

      This test examines the efficiency of record object
      manipulation, in particular, the cpu time taken in
      component selection. Both simple and variant records are
      used which are declared on the stack, on the heap and in
      library packages. The records contain scalar components.

      Record Component Selection:
```

Record Type	On-stack Record	Library Record	Heap Record
Simple	32.5us	40.2us	99.1us
Variant	51.5us	52.6us	74.8us

```
I.2.  TI01B

      This test examines the efficiency of record object
      manipulation, in particular, the cpu time taken in
      component selection. Both simple and variant records are
      used which are declared on the stack, on the heap and in
      library packages. The records contain record components.


      Record Component Selection:
```

Record Type	On-stack Record	Library Record	Heap Record
Simple	32.5us	40.2us	100us
Variant	155us	155us	290us

I.3. TI01C

This test examines the efficiency of record object manipulation, in particular, the cpu time taken in record assignment. Both simple and variant records are used which are declared on the stack, on the heap and in library packages. The records contain both scalar and record components, the variant records containing two or three discriminants.

Record Assignment:

Record Type	On-stack Record	Library Record	Heap Record
Simple	279us	286us	293us
Variant	1.59ms	1.61ms	1.96ms

I.4. TI01D

This test examines the efficiency of record object manipulation, in particular, the cpu time taken in record comparison. Both simple and variant records are used which are declared on the stack, on the heap and in library packages. The records contain both scalar and record components, the variant records containing two or three discriminants.

Record Comparison:

Record Type	On-stack Record	Library Record	Heap Record
Simple	410ns	410ns	410ns
Variant	93.3us	77.5us	250us

I.5. TI02A

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in component indexing. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages. One-, two- and three-dimensional character arrays are used.

Array Component Indexing:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	664us	961us	977us
Packed	664us	890us	665us

I.6. TI02B

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in array assignments. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages. One-, two- and three-dimensional character arrays are used.

Array Assignment:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	168us	170us	182us
Packed	168us	170us	183us

I.7. TI02C

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in array comparisons. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages. One-, two- and three-dimensional character arrays are used.

Character Array Comparison:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	410ns	410ns	410ns
Packed	410ns	410ns	410ns

I.8. TI02D

This test examines the efficiency of array object manipulation, in particular, the cpu time taken in performing logical operations on boolean arrays. Timings are taken for both packed and unpacked arrays, declared on the stack, on the heap and in library packages.

Logical Operations on Boolean Arrays:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	587us	628us	805us
Packed	196us	203us	231us

I.9. TI02E

This test examines the efficiency of array object
manipulation, in particular, the cpu time taken in array
concatenation. Timings are taken for both packed and
unpacked arrays, declared on the stack, on the heap and
in library packages. An unconstrained one-dimensional
integer array is used.

Test failed. Wrong TEST_ID in TEST.TST

I.10. TI02F

This test examines the efficiency of array object
manipulation, in particular, the cpu time taken in array
slicing. Timings are taken for both packed and unpacked
arrays, declared on the stack, on the heap and in library
packages. An unconstrained one-dimensional integer array
is used.

Array Slicing:

Array Type	On-stack Array	Library Array	Heap Array
Unpacked	566us	573us	1.48ms
Packed	584us	573us	1.48ms

Observation 2: AES Storage management tests for heap storage creep and fragmentation.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Group M, Tests TM06-TM07.

M. Group M - Storage Management Tests

M.7. TM05

This test examines the creeping of heap storage space
when returning unconstrained types from subprograms.

"Creeping" loss of storage can occur when subprograms
return large unconstrained arrays or records. This is
because it may be difficult for the runtime System to
monitor and later reclaim this space.

Returning unconstrained records does cause "creeping"
loss of heap storage.

Returning unconstrained arrays does cause "creeping" loss
of heap storage.

M.9. TM07

This test checks for fragmentation of heap storage.

The test checks whether the space recovery mechanism

merges adjacent areas of free memory, splits previously
allocated large areas into small ones, or otherwise
fragments the heap storage area.

The whole of heap storage was allocated using large
records, deallocated, allocated sing small records,
deallocated, then reallocated with the same large
records. The total amount of space allocated in each
case, and the maximum allocatable record size, was
compared. Fragmentation of heap storage does occur.

Observation 3: ACEC test of performance variation due to location of the data declaration.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Reference Variables Defined in Different Packages."

```
-------------------------------------------------------------------

                        Language Feature Overhead

-------------------------------------------------------------------
Reference Variables Defined In Different Packages
-------------------------------------------------------------------
Test        Execution   Bar                                 Similar
Name        Time        Chart                               Groups
-------------------------------------------------------------------
ss469         0.78      *                                 |
ss470         0.78      *                                 |
ss476         1.56      *                                  |
ss471         1.68      *                                   |
ss474         2.52      **                                   |
ss475         2.70      **                                    |
ss477        10.90      *********                              |
ss472        13.30      ***********                             |
ss473        42.30      ********************************         |
-------------------------------------------------------------------


-------------------------------------------------------------------
                        Individual Test Descriptions
-------------------------------------------------------------------
-- explore overheads necessary to maintain addressability
-------------------------------------------------------------------
ss469 p0467t01.a := p0467t02.d ;
-- reference variable defined in two packages
-------------------------------------------------------------------
ss470 p0467t01.a := p0467t01.b ;
-- reference variable defined in 1 external packages
-------------------------------------------------------------------
ss471 p0467t01.c ( local_one ) := local_one + p0467t02.d + global.ten ;
-- reference variable defined in local scope plus 1 -- external package
-------------------------------------------------------------------
ss472 FOR i IN p0467t01.c'RANGE LOOP
         p0467t01.c(i) := i + p0467t02.d + global.ten ;
      END LOOP ;
-- reference variable defined in three different packages
-------------------------------------------------------------------
ss473 FOR i IN p0467t01.c'RANGE Loop
         p0467t01.c(i) := i + p0467t02.d + global.ten ;
         proc0 ;
      END LOOP ;
-- reference variable defined in four different packages
```

```
-----------------------------------------------------------------
ss474 ii := p0467t01.a + p0467t02.e ;
           p0467t01.c(1) := p0467t01.b + p0467t02.f + ei ;
-- reference variable defined in three different packages
-- Multiple references to packages so might share addressing setup.
-----------------------------------------------------------------
ss475 p0467t03.g ( global.ei ) :=
           1 - p0467t01.a + p0467t02.e + p0467t03.i - local_one ;
-- reference variable defined in two different packages
-- Reference one package twice.
-----------------------------------------------------------------
ss476 p0467t03.h ( ei ) := ten ;
-- reference variable defined in two different packages
-----------------------------------------------------------------
ss477 FOR i IN int'(1)..int'(10) LOOP
         p0467t03.g ( i ) := p0467t03.h ( i ) ;
      END LOOP ;
-- reference variable defined in one external package
-----------------------------------------------------------------
```

Observation 4: ACEC test for performance variation based on order of declaration.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Reference 0th..1024th Real Variable in Package."

```
-----------------------------------------------------------------

                        Language Feature Overhead

-----------------------------------------------------------------
Reference 0th .. 1024th Real Variable In Package
-----------------------------------------------------------------
Test      Execution    Bar                                Similar
Name      Time         Chart                              Groups
-----------------------------------------------------------------
ss785       1.55       **********************************  |
ss779       1.56       **********************************  |
ss780       1.56       **********************************  |
ss781       1.56       **********************************  |
ss782       1.56       **********************************  |
ss783       1.56       **********************************  |
ss784       1.56       **********************************  |
ss786       1.56       **********************************  |
ss787       1.56       **********************************  |
ss788       1.56       **********************************  |
-----------------------------------------------------------------


-----------------------------------------------------------------
                        Individual Test Descriptions
-----------------------------------------------------------------
-- Early variables may be able to use short displacements.
-----------------------------------------------------------------
ss779 xx := r0 ;    r0 := yy ;
-- Reference the 0th real variable declared in a package.
-----------------------------------------------------------------
ss780 xx := r2 ;    r2 := yy ;
-- Reference the 2ed real variable declared in a package.
-----------------------------------------------------------------
ss781 xx := r8 ;    r8 := yy ;
-- Reference the 8th real variable declared in a package.
-----------------------------------------------------------------
```

```
ss782 xx := r16 ;    r16 := yy ;
-- Reference the 16th real variable declared in a package.
-----------------------------------------------------------
ss783 xx := r32 ;    r32 := yy ;
-- Reference the 32th real variable declared in a package.
-----------------------------------------------------------
ss784 xx := r64 ;    r64 := yy ;
-- Reference the 64th real variable declared in a package.
-----------------------------------------------------------
ss785 xx := r128 ;    r128 := yy ;
-- Reference the 128th real variable declared in a package.
-----------------------------------------------------------
ss786 xx := r256 ;    r256 := yy ;
-- Reference the 256th real variable declared in a package.
-----------------------------------------------------------
ss787 xx := r512 ;    r512 := yy ;
-- Reference the 512th real variable declared in a package.
-----------------------------------------------------------
ss788 xx := r1024 ;    r1024 := yy ;
-- Reference the 1024th real variable declared in a package.
-----------------------------------------------------------
```

References

* none

2.5 Enumeration Types

Question: How does the performance of operations on objects of an enumeration type compare with the performance of an equivalent representation using strings or numeric values?

Summary: When integer values are used as an alternative to literals of an enumeration type, there is no performance difference. For a simple assignment of a literal value, the integer assignment is slower by about 16%. (In the benchmark suites considered in this document, there are no tests that use character or string values as an alternative.) When an enumeration representation clause is used to specify the internal codes of the literals of an enumeration type, the performance of the VAL, POS, and SUCC attributes degrades by almost an order of magnitude.

Discussion: Enumeration types provide users with a versatile way of expanding the set of types that characterize their applications. They allow users to define new discrete data types that go beyond Ada's basic predefined discrete types. An issue that arises naturally when new data types are used is whether or not the performance of operations on these types is better or worse than the performance of an equivalent representation using the basic discrete types.

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

    Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

    Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: ACEC tests of assignment to an enumerated type.

In this observation, and in all that follow, the results presented are for ACEC tests; there are no equivalent tests in the AES or PIWG suites.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Range Constraint Check."

```
-------------------------------------------------------------------------

                          Optimizations

-------------------------------------------------------------------------
Range Constraint Check
-------------------------------------------------------------------------
Test            Execution    Bar                              Similar
Name            Time         Chart                            Groups
-------------------------------------------------------------------------
ss128             2.51       ********************************    |
ss255             2.51       ********************************    |
ss129             2.58       ********************************    |
```

```
-----------------------------------------------------------------------
                    Individual Test Descriptions
-----------------------------------------------------------------------
TYPE color IS ( white, red, yellow, green, blue, brown, black ) ;
hue     : color := yellow ;
------- --------------------------------------------------------
ss128 IF hue < black THEN hue := color'succ ( hue ) ; END IF ;
      IF hue > white THEN hue := color'pred ( hue ) ; END IF ;
   -- uses 'SUCC and 'PRED on enumerated type, no checking
-----------------------------------------------------------------------
ss129 IF ei < 6 THEN ei := ei + 1 ; END IF ;
      IF ei > 0 THEN ei := ei - 1 ; END IF ;
   -- Same computations as in ss128 on integers .
-----------------------------------------------------------------------
ss255 IF hue < black THEN hue := color'succ ( hue ) ; END IF ;
      IF hue > white THEN hue := color'pred ( hue ) ; END IF ;
   -- uses 'SUCC and 'PRED on enumerated type, enabling range checking
-----------------------------------------------------------------------
```

Observation 2: Assignments to array elements of enumeration types. The tests of interest are ss53 and ss309.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Arrays - Assignment to 1 Dimensional Array of Real."

```
-----------------------------------------------------------------------

                    Language Feature Overhead

-----------------------------------------------------------------------
Arrays - Assignment To 1 Dimensional Array Of Real
-----------------------------------------------------------------------
Test      Execution    Bar                              Similar
Name      Time         Chart                            Groups
-----------------------------------------------------------------------
ss55      0.78         ****                                |
ss169     0.78         ****                                |
ss645     1.28         *******                           |
ss53      1.28         *******                           |
ss309     1.28         *******                           |
ss54      1.42         ********                        |
ss646     3.95         *********************           |
ss647     6.26         **********************************   |
-----------------------------------------------------------------------
                    Individual Test Descriptions
-----------------------------------------------------------------------

-----------------------------------------------------------------------
ss53   ii := i1 ( ei ) ;
   -- Reference to subscripted array of int, without checking.
-----------------------------------------------------------------------
ss54   ii := i1 ( ei + 1 ) ;
   -- Reference to subscripted array of int, without checking.
-----------------------------------------------------------------------
ss55   ii := i1 ( 1 ) ;
   -- Reference array with a constant subscript, without checking.
-----------------------------------------------------------------------
ss169  ii := i1 ( 1 ) ;
   -- fetch from 1D array with range checking, using constant subscript
-----------------------------------------------------------------------
ss173  ii := i1 ( ei + 1 ) ;
   -- Reference to subscripted array of int, with checking.
```

```
------------------------------------------------------------
ss309  hue := stat ( ei ) ;
   -- access array of an enumerated type
------------------------------------------------------------
ss645  one := e1 ( ei ) ;
   -- fetch from 1D array, checking suppressed
------------------------------------------------------------
ss646  one := e2 ( ei , ej ) ;
   -- fetch from 2D array, checking suppressed
------------------------------------------------------------
ss647  one := e3 ( ei , ej , ek ) ;
   -- fetch from 3D array, checking suppressed
------------------------------------------------------------
```

Observation 3: Assignment of a literal value to an enumeration type.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Assign Enumeration Literal."

```
------------------------------------------------------------

                     Language Feature Overhead

------------------------------------------------------------
Assign Enumeration Literal
------------------------------------------------------------
Test      Execution   Bar                                Similar
Name      Time        Chart                              Groups
------------------------------------------------------------
ss310       0.66      *****************************     |
ss7         0.77      *********************************  |
------------------------------------------------------------
                     Individual Test Descriptions
------------------------------------------------------------
------------------------------------------------------------
ss7   kk := 1 ;
-- Integer literal assignment, literal "1" to library scope variable.
------------------------------------------------------------
ss310 hue := yellow ;
-- assign enumeration literal to variable of type
------------------------------------------------------------
```

Observation 4: The results below show the effects of forcing a particular implementation of the enumeration type by using an enumeration representation clause.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Attributes on Enumeration Types."

```
------------------------------------------------------------

                     Language Feature Overhead

------------------------------------------------------------
Attributes On Enumeration Types
------------------------------------------------------------
Test      Execution   Bar                                Similar
```

```
Name       Time      Chart                                        Groups
---------------------------------------------------------------------------
ss252      3.51      ***                                           |
ss251      5.53      *****                                         |
ss254      33.20     *********************************             |
ss253      34.60     ***********************************           |
---------------------------------------------------------------------------
                        Individual Test Descriptions
---------------------------------------------------------------------------
ss253 and ss254 use a representation clause, ss251 and ss252 do not.

TYPE mx1 IS ( add, sub, mul, lda, sta, stz ) ;     -- from LRM 13.3
a : mx1 ;
d : int := mx1'pos ( lda ) - mx1'pos ( mul ) ;
----------------------------------------------------------------
ss251 a := mx1'val ( mx1'pos ( mx1'succ ( a ) ) * ei - d ) ;
-- use VAL, POS, SUCC attributes on enumeration type without
-- representation clauses.  This statement enables range checking.
----------------------------------------------------------------
ss252 a := mx1'val ( mx1'pos ( mx1'succ ( a ) ) * ei - d ) ;
-- use VAL, POS, SUCC attributes on enumeration type without
-- representation clauses in a block with suppress RANGE_CHECK.
----------------------------------------------------------------
TYPE mx2 IS ( add, sub, mul, lda, sta, stz ) ;     -  from LRM 13.3
FOR mx2 USE ( 1, 2, 3, 8, 24, 33 ) ;
a : mx2 ;
d : int := mx2'pos ( lda ) - mx2'pos ( mul ) ;
----------------------------------------------------------------
ss253 a := mx2'val ( mx2'pos ( mx2'succ ( a ) ) * ei - d ) ;
-- use VAL, POS, SUCC attributes on enumeration type with
-- representation clause and enable range checking
----------------------------------------------------------------
ss254 a := mx2'val ( mx2'pos ( mx2'succ ( a ) ) * ei - d ) ;
-- use VAL, POS, SUCC attributes on enumeration type with
-- representation clauses,  suppressing range_checking
---------------------------------------------------------------------------
```

References

* none

2.6 Exceptions

Question: What are the performance consequences of providing exception handling capabilities?

Summary: The presence of an exception handler that is not invoked does not add to the execution time of a program. Raising and handling an exception locally takes approximately 68 to 71 microseconds. Propagating an exception through two to four levels of nesting takes from 124 microseconds to 292 microseconds, i.e., an increase in execution time ranging from 77 percent to 317 percent.

Discussion: Section 14.5.4 of the Rationale for the Design of the Ada Programming Language states that, "One important design consideration for the exception handling facility is that exceptions should add to execution time only if they are raised." The benchmark results presented in the Observation sections that follow support this assertion for the configuration tested. Note that, although the tests facilitate comparisons between the various exception-handling schemes, they do not provide any comparison between exception-handling and alternatives to exception-handling such as the use of status flags or return variables.

There are some inconsistencies in the results presented here. The ACEC and PIWG tests to raise and handle an exception locally yield the same result (about 70 microseconds), but the corresponding AES test yields a result more than twice as large (170 microseconds). The ACEC result for exception handling during rendezvous (306 microseconds) is about 38% larger than the PIWG result (221 microseconds).

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

    Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

    Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: ACEC and PIWG tests that raise and handle locally a user-defined exception.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "User-Defined Exceptions" and "Exceptions—Conditional Raise."

In this group of results, subtracting the results for tests ss311 and ss312 yields 71 microseconds, approximately, for the time required to raise the exception and enter the handler. A similar calculation for ss312 and ss313 shows that the presence of a handler that is not entered adds only 0.26 microseconds of overhead.

```
                        Runtime System Behavior

--------------------------------------------------------------------------
User-defined Exception
--------------------------------------------------------------------------
Test        Execution    Bar                                   Similar
Name        Time         Chart                                 Groups
--------------------------------------------------------------------------
ss313         0.52                                              |
ss312         0.78                                              |
ss311        71.30       ********************************       |
--------------------------------------------------------------------------
                       Individual Test Descriptions
--------------------------------------------------------------------------
Demonstrate overhead of raising user-defined exceptions.
-----------------------------------------------------------------------
ss311 DECLARE
          except : EXCEPTION ;
      BEGIN
        IF l1 > 0
        THEN
            RAISE except ;
        END IF ;         -- True, raised
      EXCEPTION
        WHEN except => proc0 ;
      END ;
-- explicit raise of user-defined exception, and process it
-----------------------------------------------------------------------
ss312 DECLARE
          except : EXCEPTION ;
      BEGIN
        IF l1 < 0
        THEN
            RAISE except ;
        END IF ;         -- False, not raised
      EXCEPTION
        WHEN except => proc0 ;
      END ;
-- define user-defined exception, do not raise it
-----------------------------------------------------------------------
ss313 DECLARE
      BEGIN
        IF l1 < 0
        THEN
            RAISE Numeric_error ;
        END IF ;         -- False, not raised
      END ;
-- does not define an exception or raise one
-----------------------------------------------------------------------
```

This second group of ACEC tests also shows that raising an exception and entering the handler takes approximately 71 microseconds.

```
--------------------------------------------------------------------------

                        Runtime System Behavior

--------------------------------------------------------------------------
Exception - Conditional Raise
--------------------------------------------------------------------------
Test        Execution    Bar                                   Similar
```

```
Name        Time        Chart                                       Groups
--------------------------------------------------------------------------------
ss527       5.73        ***                                         |
ss528       76.60       **********************************          |
------.    ---------------------------------------------------------------------
                        Individual Test Descriptions
--------------------------------------------------------------------------------
ss527 DECLARE
            ex : EXCEPTION ;
        BEGIN
          proc0 ;
          IF mm = 11  -- never taken
          THEN
              RAISE ex ;
          END IF ;
          proc0;
        EXCEPTION
          WHEN ex => proc0;
        END ;
-- conditional raise of user defined exception and go through handler.
-- Not taken.  Compare with ss528, where exception is raised.
------------------------------------------------------------------------
ss528 DECLARE
            ex : EXCEPTION ;
        BEGIN
          proc0 ;
          IF mm /= 11 -- always taken
          THEN
              RAISE ex ;
          END IF ;
          proc0;
        EXCEPTION
          WHEN ex => proc0;
        END ;
-- conditional raise user defined exception and go through
-- handler. Taken. Contrast with ss527 where it is not.
-- Explicit RAISE could be implemented by simple branch.
------------------------------------------------------------------------
```

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Test E000001.

```
Test Name:    E000001                     Class Name:  Exception
CPU Time:        68.4  microseconds
Wall Time:       68.5  microseconds.       Iteration Count:   256
Test Description:
 Time to raise and handle an exception
 Exception defined locally and handled locally
```

Observation 2: ACEC and PIWG tests that raise an exception in a called procedure and handle it in the calling unit.

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Report "Exceptions - Block With Handler."

Subtracting the ss384 result from the ss381 results yields a value of, approximately, 117 microseconds.

```
                 -----------------------------------------------------------------------

                                      Runtime System Behavior

                 -----------------------------------------------------------------------


                 -----------------------------------------------------------------------
                 Exception - Block With Handler
                 -----------------------------------------------------------------------
                 Test        Execution    Bar                                    Similar
                 Name        Time         Chart                                  Groups
                 -----------------------------------------------------------------------
                 ss384          6.46     **                                        |
                 ss381        123.90     ***********************************       |
                 -----------------------------------------------------------------------
                                      Individual Test Descriptions
                 -----------------------------------------------------------------------

                 -------------------------------------------------------------
                 ss381 DECLARE
                       BEGIN
                          xx := one ;
                          f ;
                       EXCEPTION
                          WHEN excp => NULL ;
                          WHEN OTHERS => die ;
                       END ;
                 -- Block with exception handler which calls on a procedure
                 -- which raises the exception (the procedure it calls
                 -- on does not have a handler but simply raises the exception.)
                 -------------------------------------------------------------
                 ss384 DECLARE
                       BEGIN
                          xx := one ;
                          f ;
                       EXCEPTION
                          WHEN excp => NULL ;
                          WHEN OTHERS => die ;
                       END;
                 -- call on procedure which doesn't propagate exception
                 -------------------------------------------------------------
```

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Test E000002.

The 7% increase in this result over the equivalent ACEC result just listed is not significant. According to the comments in this test, the difference between the PIWG result shown here and the PIWG E000001 result shown in the previous observation is equal to the pure propagation overhead. Performing the subtraction yields a value of, approximately, 56 microseconds. This is consistent with the result of a similar AES test shown in Observation 4.

```
       Test Name:   E000002                  Class Name:  Exception
       CPU Time:       124.7  microseconds
       Wall Time:      124.7  microseconds.   Iteration Count:    128
       Test Description:
        Exception raise and handle timing measurement
        when exception is in a procedure in a package
```

Observation 3: ACEC and PIWG tests of exception propagation.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Exception Processing with/without Handlers."

The results below show the overhead of propagating an exception through various numbers of levels for eventual handling. This set of tests confirms the increasing performance penalty for propagating an exception through increasingly nested units.

As before the results are obtained by subtraction of the results from pairs of tests. The result for ss381 minus ss384 has already been covered in Observation 2. Subtracting the ss383 result from that of ss380 yields a time of approximately 161 microseconds. A similar calculation for ss382 and ss379 yields 227 microseconds, approximately. These latter two tests are not directly comparable with the PIWG test results shown in this observation because of the different numbers of levels through which the exceptions are propagated.

```
    --------------------------------------------------------------------

                        Runtime System Behavior

    --------------------------------------------------------------------
    Exceptions Processing with/without Handlers
    --------------------------------------------------------------------
    Test      Execution   Bar                                 Similar
    Name        Time      Chart                               Groups
    --------------------------------------------------------------------
    ss384         6.46    *                                      |
    ss383        13.60    **                                     |
    ss382        14.40    **                                      |
    ss381       123.90    *****************                       |
    ss380       174.40    **************************               |
    ss379       241.10    **********************************        |
    --------------------------------------------------------------------

    --------------------------------------------------------------------
                        Individual Test Descriptions
    --------------------------------------------------------------------
    PROCEDURE h1 IS
    BEGIN
        g1 ;
    EXCEPTION
        WHEN excp   => NULL ;
        WHEN OTHERS => die ;
    END h1 ;
    ----------------------------------------------------------
    ss379 h1  ;    -- does propagate EXCEPTION
    -- make two PROCEDURE calls. The lowest level has an EXCEPTION handler
    -- which can (re) raise an EXCEPTION and propagate it to the next
    -- higher level. this problem raises the EXCEPTION.
    ----------------------------------------------------------
    ss382 h1  ;    -- EXCEPTION not propagated
    -- make two PROCEDURE calls. The lowest level has an EXCEPTION handler
    -- which can (re) raise an EXCEPTION and propagate it to the next
    -- higher level. this problem does NOT raise the EXCEPTION.
    ----------------------------------------------------------
    PROCEDURE h2 IS
    BEGIN
        g2 ;
    EXCEPTION
        WHEN excp   => NULL ;
```

```
          WHEN OTHERS => die ;
      END h2 ;
      --------------------------------------------------------------
      ss380 h2  ;    -- raises EXCEPTION
      -- make two PROCEDURE calls. The lowest level does not have an
      -- EXCEPTION handler and will simply propagate the EXCEPTION raised
      -- to the next higher level.  this problem raises the EXCEPTION.
      ---------------------------------------------------------------
      ss383 h2  ;    -- EXCEPTION not raised
      -- make two PROCEDURE calls. The lowest level does not have an
      -- EXCEPTION handler and will simply propagate the EXCEPTION raised to
      -- the next higher level.  this problem does not raise the EXCEPTION.
      ---------------------------------------------------------------
      PROCEDURE f IS
      BEGIN
         proc0 ;
         IF ii > 0
         THEN
            RAISE excp ;
         END IF ;
      END f ;
      --------------------------------------------------------------
      ss381 DECLARE
            BEGIN
               xx := one ;
               f ;
            EXCEPTION
               WHEN excp   => NULL ;
               WHEN OTHERS => die ;
            END ;
      -- Block with exception handler which calls on a procedure which
      -- raises the exception (the procedure it calls on does not have
      -- a handler but simply raises the exception.)
      -------------------------------------------------------------
      ss384 -- the same as ss381, except that the exception is not raised
      -- call on procedure which doesn't propagate exception
      ---------------------------------------------------------------------
```

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Tests E000003 and E000004.

```
      Test Name:    E000003                    Class Name:  Exception
      CPU Time:       235.5  microseconds
      Wall Time:      235.5  microseconds.      Iteration Count:   64
      Test Description:
       Exception raise and handle timing measurement
       when exception is raised nested 3 deep in procedure calls

      Test Name:    E000004                    Class Name:  Exception
      CPU Time:       291.9  microseconds
      Wall Time:      292.0  microseconds.      Iteration Count:   64
      Test Description:
       Exception raise and handle timing measurement
       when exception is nested 4 deep in procedures
```

Observation 4: ACEC and PIWG tests of exception-raising during task rendezvous.

The rules of Ada state that when an exception is raised within an accept statement (and not handled in an inner frame,) the same exception is raised again in the called task, immediately after the accept statement, and is also propagated to the calling task.

ACEC Test Results:

Configuration 1, ACEC Release 2.0, Raw Output.

There is no set of SSA tests for this form of exception, but two tasking tests from the main ACEC suite, task37a and task37b, show that the overhead is (subtracting the task37b result from that of task37a), approximately, 306 microseconds.

```
                                          outer loop count
                                    inner loop count  |
                                                   |  |
    name                         size      min      mean  |  |  sigma

    null loop_time                  0 1.8086E+00                 0.1%
    task37a
      -- Raises a user-defined exception inside a rendezvous.
      -- The exception will be propagated to task performing ENTRY call.
      -- cf. task37b
                                   48     2050.1    2055.0  4  3  0.2%
    task37b
      -- This test problem was constructed for comparison with TASK37A
      -- Unlike TASK37A, this test problem rendezvous and terminates the
      -- task without raising and propagating an exception.
      -- Difference between TASK37B and TASK37A is the incremental time
      -- to raise and propagate an exception from within a rendezvous.
                                   48     1744.0    1744.0  4  3  0.0%
```

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Test E000006.

There is a considerable discrepancy between the ACEC result (306 microseconds) and the PIWG result (221 microseconds). There is no AES test to measure the overhead of an exception occurring during a rendezvous.

```
Test Name:   E000005                     Class Name:  Exception
CPU Time:       221.3  microseconds
Wall Time:      220.9  microseconds.     Iteration Count:   32
Test Description:
  Exception raise and handle timing measurement
  when exception is in a rendezvous
  Both the task and the caller must handle the exception
```

Observation 5: AES exception handling tests.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Test TI12.

The AES exception-handling test combines a number of tests in a single program. In the final test, a recursive scheme is used to measure the time to raise and handle an exception that is propagated through several levels. From these multi-level measurements, the average time to propagate an exception through just one level is obtained. By comparing the 5-level and 10-level averages shown, it is possible to determine if the single-level propagation time is invariant or is dependent upon the number of levels. The 56-microsecond result for this test is consistent with the PIWG result obtained by subtracting the PIWG E000001 and E000002 results, as discussed in Observation 2. However, the 170-microsecond time to raise and handle an exception locally is more than twice the ACEC and PIWG results for the same test.

There is no AES test to measure the overhead of an exception occurring during a rendezvous.

```
I. Group I - runtime Efficiency Tests


I.27. TI12

        This test examines the runtime efficiency  of  exception
        handling.

        The  cpu  time  taken to raise an exception using a RAISE
        statement, and handle it in the same block is 170us.

        The  additional  cpu  entry  and  exit  overhead  for   a
        subprogram with an exception handler is 0s.

        The   stack   space   overhead  of  calling  a  procedure
        containing an exception handler, compared with calling  a
        procedure  without  an  exception  handler  is   0
        STORAGE_UNITs.

             +--------+---------+--------------+
             | No. of | Nesting | Cpu time for |
             | Levels |  Depth  | Propagation  |
             +--------+---------+--------------+
             |    5   |    7    |    55.6us    |
             |   10   |   12    |    55.8us    |
             +--------+---------+--------------+
```

References

[Ichbiah]   Ichbiah, Barnes, Firth, Woodger. *Rationale for the Design of the Ada Programming Language.* United States Government, 1986.

2.7 Generic Units

Question: What is the comparative performance of generic and non-generic units?

Summary: Efficiency of generic units is comparable to non-generic units. For the cases tested for this report, a conservative assumption is that a generic unit will increase execution time by 5%. In many cases the generic unit's performance was equal to or exceeded the comparable non-generic version. Code sharing did not occur.

Discussion: Generic program units allow a single, general version of software to be used for different data types by the process of instantiation. Generic units offer several possible advantages. An algorithm can be implemented once and instantiated for different data types, simplifying program construction and maintenance. In some cases, code sharing may reduce the storage requirements for executing code by reducing the number of separate units. However, reduced execution speed is often assumed for generics of all kinds, which argues against their use.

Comparison of matched generic and non-generic procedures indicate that the generic routines are generally somewhat slower than their custom-coded counterparts (Observation 2 on page 56, Observation 3 on page 58; and Observation 4 on page 59). Generic versions execution speed ranged from -35% to +6% of the equivalent non-generic routines. The lower execution time noted for generics in Observation 2 is in comparison to "hand-coded" equivalents, and probably represents inefficient hand coding rather than true improvement due to the use of generics. The effects of generic instantiation vary depending on the application code however, so these numbers are, at best, representative.

If multiple instantiations of generic routines can share an executable image, storage is conserved compared to multiple hand-coded routines. Tests demonstrate that code sharing is not performed by the tested VADS compiler version (Observation 1 on page 55).

The effect of inlining generic routines (Observation 4 on page 59) is ambiguous. For the small number of tested cases, conflicting results are noted. The effects are small however, so inlining is not considered harmful, although beneficial effects have not been conclusively demonstrated.

Configuration(s) tested:

1.   **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

     Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

     Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: The AES test TO09 checks explicitly for code sharing. No code sharing was observed

---

for any of the test scenarios.

(See also the AES tests TI09A-G, which show no code sharing. TI09 output is below in Observation 2 on page 56.)

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Test TO09.

```
O. Group O - Optimisation Tests

O.9.  TO09

       This test checks on the sharing of code by generic
       bodies.

       Simple generic procedures tak  g discrete input and
       output parameters which are  instantiated in the same
       compilation unit do not share code.

       Simple parameterless generic packages which are
       instantiated in the same compilation unit do not share
       code.

       Simple generic packages taking a discrete parameter which
       are instantiated in the same compilation unit do not
       share code.

       Simple parameterless generic packages which are declared
       as individual compilation units do not share code.

       Generic packages taking a number of generic parameters
       which are declared as individual compilation units do not
       share code.
```

Observation 2: AES tests TI09A-G compare the performance of generic routines and hand-coded equivalents. Performance of the generic routines varied from 4.5% slower to 35.3% faster. The reported increase of 35% for TG09F, while impressive, may be an artifact of a low performance hand-coded version.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Tests TI09A-G.

```
I. Group I - Runtime Efficiency Tests

I.18.  TI09A

       This test assesses the relative efficiency of passing and
       using an enumeration type as a generic parameter against
       its non-generic equivalent. The test also measures the
       cpu time taken to perform the generic package. The test
       determines whether or not instantiations of the same
       generic package are able to share the code of their
       bodies.

                    Package instantiation    : 23.7us
                    Package execution        : 1.18ms
                    Execution of handed-coded : 1.09ms
```

equivalent

Separate instantiations of the package do not share code.

I.19. TI09B

This test assesses the relative efficiency of passing and
using an array type as a generic parameter against its
non-generic equivalent. The test also measures the cpu
time taken to perform the generic package. The test
determines whether or not instantiations of the same
generic package are able to share the code of their
bodies.

           Package instantiation   : 23.3us
           Package execution   : 299ms
           Execution of handed-coded : 313ms
               equivalent

Separate instantiations of the package do not share code.

I.20. TI09C

This test assesses the relative efficiency of passing and
using a fixed point type as a generic parameter against
its non-generic equivalent. The test also measures the
cpu time taken to perform the generic package. The test
determines whether or not instantiations of the same
generic package are able to share the code of their
bodies.

Test failed. !STORAGE_ERROR

I.21. TI09D

This test assesses the relative efficiency of passing and
using a floating point type as a generic parameter
against its non-generic equivalent. The test also
measures the cpu time taken to perform the generic
package. The test determines whether or not
instantiations of the same generic package are able to
share the code of their bodies.

Test failed. !STORAGE_ERROR

I.22. TI09E

This test assesses the relative efficiency of passing and
using a record type as a generic parameter against its
non-generic equivalent. The test also measures the cpu
time taken to perform the generic package. The test
determines whether or not instantiations of the same
generic package are able to share the code of their
bodies.

           Package instantiation   : 1.01ms
           Package execution   : 3.10ms
           Execution of handed-coded : 3.09ms
               equivalent

Separate instantiations of the package do not share code.

I.23. TI09F

This test assesses the relative efficiency of passing and using a discriminated record type as a generic parameter against its non-generic equivalent. The test also measures the cpu time taken to perform the generic package. The test determines whether or not instantiations of the same generic package are able to share the code of their bodies.

Package instantiation   : 2.63ms
Package execution   : 3.33ms
Execution of handed-coded : 5.07ms
    equivalent

Separate instantiations of the package do not share code.

I.24. TI09G

This test assesses the relative efficiency of passing and using a function as a generic parameter against its non-generic equivalent. The test also measures the cpu time taken to perform the generic package. The test determines whether or not instantiations of the same generic package are able to share the code of their bodies.

Package instantiation   : 1.76ms
Package execution   : 2.58ms
Execution of handed-coded : 2.57ms
    equivalent

Separate instantiations of the package do not share code.

Observation 3: AES use of the GAMM standard shows the performance of a generic instantiation to be identical to that of the non-generic version (TJ01 vs. TJ07).

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Tests TJ01 and TJ07.

J. Group J - NPL Test Suite

J.1. TJ01

This benchmark test determines the cpu time taken to perform a set of standard scientific calculations. This is known as the GAMM standard.

The GAMM standard is 4.95us.

J.7. TJ07

This benchmark test measures the change in the GAMM

```
standard caused by using generic instantiations.

There was a 0% increase in the GAMM standard.

Note that the results of this test may be  imprecise  as
extraneous influences were present.
```

Observation 4: The two ACEC SSA reports titled "Generic Function Calls - Inline vs. Non-Inline" examine the behavior of a simple maximum function which is implemented in generic and regular forms, subjected to inlining and placed at several locations (local, same unit, and external unit). In one test group, the test makes a single call to the function. In the second, two calls are made, with the return value of the first call used as a parameter for the second.

The simple function call (one call, no nesting) shows that generic performance ranges from equivalent to the non-generic forms to somewhat worse. When two calls are nested, the non-generic forms (ss632 and ss143) show the best performance while generic execution time increases from 4 to 6%.

Inlining combined with generics shows mixed results.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Reports "Language Feature Overhead"

"Generic Function Calls - Inline Vs. Non-inline" [tests ss141, ss142, ss621, ss622, ss623, ss624, ss625 and 22626]

"Generic Function Calls - Inline Vs. Non-inline" [tests ss143, ss628, ss629, ss630, ss631, ss632 and ss633]

Note that test ss633 simplifies the function call, and should be ignored in making comparisons. This SSA test is marked with the phrase ☞Ignore.

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------


                          Language Feature Overhead

-------------------------------------------------------------------------
Generic Function Calls - Inline Vs Non-inline
-------------------------------------------------------------------------
Test       Execution    Bar                                    Similar
Name       Time         Chart                                  Groups
-------------------------------------------------------------------------
ss142         8.19      ************************           |
ss625         8.19      ************************           |
ss141        11.10      *********************************    |
ss621        11.10      *********************************    |
ss622        11.10      *********************************    |
ss626        11.30      **********************************     |
ss624        11.60      ***********************************      |
ss623        11.80      ************************************       |
-------------------------------------------------------------------------
```

Individual Test Descriptions

```
------------------------------------------------------------------------
FUNCTION max1 ( x , y : real ) RETURN real IS
BEGIN
    IF x >= y THEN RETURN x ;
             ELSE RETURN y ;
    END IF ;
END max1 ;
------------------------------------------------------------------------
ss141 xx := max1 ( yy , zz ) ;
    -- call on local function
------------------------------------------------------------------------
ss142 xx := max2 ( yy , zz ) ;   -- max2 is inline of max1
    -- call on local inline function
------------------------------------------------------------------------
ss621   xx := max3 ( yy , zz ) ;
    -- generic non-inline function, instantiated in external unit
------------------------------------------------------------------------
ss622   xx := max4 ( yy , zz ) ;
    -- generic inline function, instantiated in external unit
------------------------------------------------------------------------
ss623   xx := max5 ( yy , zz ) ;
    -- generic non-inline function, instantiated in same unit
------------------------------------------------------------------------
ss624   xx := max6 ( yy , zz ) ;
    -- generic inline function, instantiated in same unit
------------------------------------------------------------------------
ss625   xx := max7 ( yy , zz ) ;
    -- local generic inline function,
------------------------------------------------------------------------
ss626   xx := max8 ( yy , zz ) ;
    -- local generic non-inline function,
------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
Generic Function Calls - Inline Vs Non-inline
------------------------------------------------------------------------
Test      Execution    Bar                                   Similar
Name      Time         Chart                                 Groups
------------------------------------------------------------------------
ss633     11.10        ******************          ☞Ignore
ss632     19.80        *******************************          |
ss143     20.40        ********************************         |
ss629     20.60        ********************************          |
ss630     20.60        ********************************          |
ss631     20.80        **********************************         |
ss628     21.00        **********************************          |
------------------------------------------------------------------------
```

Individual Test Descriptions

```
------------------------------------------------------------------------
FUNCTION max1 ( x , y : real ) RETURN real IS
BEGIN
    IF x >= y THEN RETURN x ;
             ELSE RETURN y ;
    END IF ;
END max1 ;
------------------------------------------------------------------------
ss143 xx := max1 ( 1.0 , max1 ( yy , zz ) ) ;
    -- local function where actual parameter contains another
------------------------------------------------------------------------
ss628   xx := max3 ( 1.0 , max3 ( yy , zz ) ) ;
    -- generic non-inline function, instantiated in external unit
------------------------------------------------------------------------
```

```
ss629    xx := max4 ( 1.0 , max4 ( yy , zz ) ) ;
    -- generic inline function, instantiated in external unit
----------------------------------------------------------------
ss630    xx := max5 ( 1.0 , max5 ( yy , zz ) ) ;
    -- generic non-inline function, instantiated in same unit
----------------------------------------------------------------
ss631    xx := max6 ( 1.0 , max6 ( yy , zz ) ) ;
    -- generic inline function, instantiated in same unit
----------------------------------------------------------------
ss632    xx := max ( 1.0 , max ( yy , zz ) ) ;
    -- language feature test comparison, non-generic non-inline function,
----------------------------------------------------------------
ss633    xx := max9 ( yy , zz ) ;
    -- language feature test comparison, inline in external package
----------------------------------------------------------------
```

References

- none

2.8 Inlining of Procedures

Question: What is the effect of inlining procedures and generic procedures?

Summary: Execution time is generally reduced by inlining, although the gain in performance varies. Inlining requests are honored by the compiler, but not in all cases. Program size changes due to inlining were not measured.

Discussion: Ada's **pragma** INLINE allows procedures to be inserted in the calling program at the point of the call, avoiding the overhead of transferring control. However, the performance gain must be balanced against the increased program size when multiple copies of routines are created.

An Ada compiler is not required to perform inlining. Likewise, inlining may be performed automatically. AES test TO05 examines conditions under which inlining is effective (Observation 1 on page 63). The AES test indicates that inlining is only performed on request, but also that the request is not always honored.

Both the AES and ACEC results indicate that **pragma** INLINE is effective in reducing execution time for ordinary (non-generic) procedures (Observation 2 on page 64 and Observation 3 on page 66). The evidence for generic procedures is less persuasive (Observation 4 on page 67), but it appears that inlining generic procedures is somewhat effective. The tests show a fair amount of variation in improved execution time. This is an expected result, since the time saved reflects the complexity of the call, affected by such factors as the number, type, and ordering of arguments and method used for passing arguments, not the complexity of the procedure body.

The size effects of **pragma** INLINE was not measured. Conceptually, the size expansion should be directly related to the amount of code the compiler produces for the body of the procedure.

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

    Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

    Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: AES test TO05 found that **pragma** INLINE is honored on demand for "simple procedures," but not for subprograms "which are difficult to inline." Inlining was not performed automatically for subprograms which did not contain **pragma** INLINE.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0, Test TO05.

```
O.5.    TO05

        This test examines subprogram inlining.

        The test determines whether pragma INLINE works under
        various conditions and also whether subprograms are
        automatically inlined (regardless of whether the pragma
        is set).  In the followin    a "simple" procedure is one
        which has several lines of code in the body but no
        exception handlers/blocks etc. and only one or two scalar
        parameters.

        Inlining is performed for simple procedures declared in
        the same compilation unit.

        Inlining is performed for simple procedures declared in a
        WITHed package.

        Inlining is not performed for simple procedures  declared
        in a WITHed package, when pragma IN.    is NOT set.

        Inlining is not performed for routines which are
        difficult to inline (e.g. one with complex parameters and
        exception handlers etc.),  though  not  unreasonably
        difficult.

        Inlining is performed for routines which pass on their
        (simple) parameters to another routine, with the addition
        of extra parameters.
```

Observation 2: AES test TI10, configuration 1, shows that performance improvements from the use of **pragma inline** are measurable when compared with subprograms that are not inlined.

Results for integer parameters appear reasonable. The remaining tests record execution times of zero seconds for the inlined routines, which invalidate the comparison to the non-inlined routines. Analysis of the test source code shows that the calling times are measured for paired procedures. The inlined version of the pairs lacks anti-optimization code present in the non-inlined version that accesses the procedure's argument. Although this anti-optimization code is protected from actual execution during testing, omitting this code allows the compiler to remove all the test code.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0, Test TI10.

Note that most of the inlining tests were removed by optimization (see above). These observations are marked with the phrase ☞**Inlining results invalid** for the following table.

I.25. TI10

This test examines the runtime efficiency of subprogram calls, in particular, the passing of scalar and non-scalar parameters using the "in", "in out", "out" and "return" modes. The test also determines whether the parameters are passed by copy or by reference. The overheads of subprogram entry and exit are included in the measured costs. The test is performed for INLINEd and non-INLINEd subprograms.

Test T005 indicates that inlining is performed for some simple procedures but not for some procedures which are fairly difficult to inline.

Integer parameters:

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	COPY	7.82us	COPY	246ns
in out	COPY	5.90us	COPY	510ns
out	COPY	5.57us	COPY	253ns
return	-	6.04us	-	-

Unconstrained array parameters (array of 256 integers):

☞ **Inlining results invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	7.69us	REF	0s
in out	REF	7.69us	REF	0s
out	REF	8.61us	REF	0s
return	-	229us	-	-

Constrained array parameters (array of 256 integers):

☞ **Inlining results invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	7.23us	REF	0s
in out	REF	6.99us	REF	0s
out	REF	6.79us	REF	0s
return	-	229us	-	-

Simple record parameters (record of 256 components):

☞**Inlining results Invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	7.37us	REF	0s
in out	REF	6.86us	REF	0s
out	REF	6.79us	REF	0s
return	-	233us	-	-

Discriminated record parameters (record of 256 components):

☞**Inlining results Invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	6.86us	REF	0s
in out	REF	6.84us	REF	0s
out	REF	6.84us	REF	0s
return	-	234us	-	-

Observation 3: ACEC Single System Analysis (SSA) report "Function Calls (Inline)" compares several variations of a procedure. Simple comparison of ss141(procedure) and ss142 (inlined procedure) shows that inlining is effective (a 26% decrease in execution time).

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Report "Language Feature Overhead" - "Function Calls (Inline)."

```
-----------------------------------------------------------------------
                     Language Feature Overhead
-----------------------------------------------------------------------
Function Calls (inline)
-----------------------------------------------------------------------
Test      Execution   Bar                                  Similar
Name      Time        Chart                                Groups
-----------------------------------------------------------------------
ss144         7.94    *************                    |
ss142         8.19    **************                      |
ss249        10.20    *****************                      |
ss141        11.10    *******************                      |
ss143        20.40    ***********************************       |
-----------------------------------------------------------------------
                     Individual Test Descriptions
-----------------------------------------------------------------------
FUNCTION max1 ( x , y : real ) RETURN real IS
BEGIN
    IF x >= y THEN RETURN x ;
             ELSE RETURN y ;
    END IF ;
```

```
END max1 ;

FUNCTION max2 ( x , y : real ) RETURN real IS
BEGIN
    IF x >= y THEN RETURN x ;
               ELSE RETURN y ;
    END IF ;
END max2 ;
PRAGMA inline ( max2 ) ;

PROCEDURE pmax ( a : OUT real ; b , c : IN real ) IS
BEGIN
    IF b >= c THEN a := b ;
               ELSE a := c ;
    END IF ;
END pmax;
------------------------------------------------------------
ss141 xx := max1 ( yy , zz ) ;
      -- call on local function
------------------------------------------------------------
ss142 xx := max2 ( yy , zz ) ;   -- max2 is inline of max1
      -- call on local inline function
------------------------------------------------------------
ss143 xx := max1 ( 1.0 , max1 ( yy , zz ) ) ;
      -- Call function where actual parameter contains another
------------------------------------------------------------
ss144 IF yy >= zz THEN xx := yy ; ELSE xx := zz ; END IF ;
      -- example of textual substitution to compare to ss142
------------------------------------------------------------
ss249 pmax ( xx , yy , zz ) ;
      -- procedure equivalent to function Max1
------------------------------------------------------------
```

Observation 4: ACEC Single System Analysis (SSA) report "Language Feature Overhead" compares inlined procedures for regular and generic procedures. The paired tests ss621 and ss622, external instantiation, (0% improvement), and ss623 and ss624, same unit, but not local (2% improvement) show slight execution time decrease for inlined generic procedures over non-inlined generic procedures. Tests ss625 and ss626, local declarations, show a larger time decrease (28% improvement) when the generic procedure is inlined. The location of the generic seems to be significant.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Language Feature Overhead" - "Generic Function Calls - Inline vs. Non-Inline."

```
--------------------------------------------------------------------------

                        Language Feature Overhead

--------------------------------------------------------------------------
Generic Function Calls - Inline Vs Non-inline
--------------------------------------------------------------------------
Test        Execution   Bar                                 Similar
Name        Time        Chart                               Groups
--------------------------------------------------------------------------
ss142           8.19    ***********************              |
ss625           8.19    ***********************              |
ss141          11.10    *********************************    |
ss621          11.10    *********************************    |
ss622          11.10    *********************************    |
```

```
ss626      11.30     **********************************          |
ss624      11.60     **********************************            |
ss623      11.80     **********************************              |
-----------------------------------------------------------------------
                      Individual Test Descriptions
-----------------------------------------------------------------------
FUNCTION max1 ( x , y : real ) RETURN real IS
BEGIN
   IF x >= y THEN RETURN x ;
             ELSE RETURN y ;
   END IF ;
END max1 ;
-----------------------------------------------------------------------
ss141 xx := max1 ( yy , zz ) ;
   -- call on local function
-----------------------------------------------------------------------
ss142 xx := max2 ( yy , zz ) ;   -- max2 is inline of max1
   -- call on local inline function
-----------------------------------------------------------------------
ss621   xx := max3 ( yy , zz ) ;
   -- generic non-inline function, instantiated in external unit
-----------------------------------------------------------------------
ss622   xx := max4 ( yy , zz ) ;
   -- generic inline function, instantiated in external unit
-----------------------------------------------------------------------
ss623   xx := max5 ( yy , zz ) ;
   -- generic non-inline function, instantiated in same unit
-----------------------------------------------------------------------
ss624   xx := max6 ( yy , zz ) ;
   -- generic inline function, instantiated in same unit
-----------------------------------------------------------------------
ss625   xx := max7 ( yy , zz ) ;
   -- local generic inline function,
-----------------------------------------------------------------------
ss626   xx := max8 ( yy , zz ) ;
   -- local generic non-inline function,
-----------------------------------------------------------------------
```

References

- none

2.9 Logical Tests

Question: What are the performance trade offs between the **case** statement and **if** statement?

Summary: Relatively little performance data is available to compare the logical statements (case and if). The available data suggest that the case statement has slightly better performance and that the compiler optimizes logical statements effectively. Coding large logical tests as tables or Boolean arrays may improve performance.

Discussion: Ada provides several statements which alter program flow based on the results of evaluating an expression. The **if** and **case** statements are commonly used to alter the flow of program execution within a task. The **if** statement provides more flexibility in defining the logical test, but the two statements are often semantically equivalent and equally clear. The application programmer therefore can select between the two statements based on considerations of clarity of expression and performance.

Primary performance considerations for logical expressions are speed of execution and the amount of storage required. Execution speed should consider both the average speed and the variation in speed.

Other considerations include:

- How does the number of alternatives affect performance of the statements?

  The algorithm being implemented may require choosing between only two alternatives or between hundreds. Performance testing should characterize the range of small to large number of alternatives.

- How does distribution of the alternatives affect performance?

  The logical test may select among closely spaced, well ordered alternatives or the alternatives may be sparsely and randomly distributed. For example, if 100 alternatives were present in the test, there may well be a performance difference when the choices range consecutively from 1 to 100 versus 100 choices which are randomly distributed between 1 to 1,000,000. The number of alternatives is the same, but the internal representation of the selection might vary. The **case** statement, in particular, can benefit from the generation of optimal selection strategies (e.g., jump table, hashed jump table, sequence of comparisons, etc.).

- Do special forms of logical expression allowed for **if** statements increase speed?

  The **case** statement limits the test expression to discrete types and the choice on the case statement alternatives (**when**) to discrete values or ranges. The **if** offers a number of variants for equivalent logical expressions. An **if** can mimic the equivalent **case** statement or use a different form that achieves the same result (for instance, by using the short circuit operators **and then** and **or else**, which have been identified as introducing performance variation).

- Does the method(s) used to nest the statements affect performance?

  When there are many branches in the logical test, **if** statements can be nested or extended without nesting using the **elsif** clause. Similarly, the **case** statement can be nested.

- Do alternative test methods, such as table look up, offer performance advantages?

  If a large or complicated branching structure is required, space or time savings achieved by programming a specific test may become important. Preparing tests tailored to the problem can use regularities in the data that the Ada compilation system might not generate automatically. The use of a table look up, indexed by the results of an expression, might be more efficient than the use of a **case** statement.

To compare the performance of logical statements, the ideal test set would compare matched sets of **case** and **If** statements, measure the range of variation as well as provide average values for the tests, and provide both time and space measures. Unfortunately, the standard benchmark suites used for this report do not provide many head-to-head comparisons between **If** and **case** statements in their current versions.

The AES SSA report compares several alternate logical tests in the two "Test For Letter Being A Vowel" entries (Observation 4 on page 75). For these tests, the **case** statement is 23% faster than the **If** statement.

Several **case** statements are timed, and the results presented in the AES report entry titled "Case Statements." The **case** statements are not comparable to each other, nor are the equivalent **If** statements provided. This entry seems mostly useful for comparing different Ada compilers.

Observations relevant to other considerations:

- How does the number of alternatives affect performance of the statements?

  The AES Test TG19 (Observation 1 on page 71) notes that "at least 5000" alternatives are allowed for **case** statements. AES Test TG32 (Observation 1 on page 71) notes that the **case** statement permits static nesting of "at least 100" deep.

  AES test TG33 (Observation 1 on page 71) notes that the **If** statement can be statically nested "at least 100" deep. The AES test TG36 (Observation 1 on page 71) notes that "at least 100" **elsif** parts are permitted.

  For each of the four tests, the VADS compiler handled the largest test case.

- How does distribution of the alternatives affect performance?

  The AES TO17 test (Observation 2 on page 72) examine how the compiler handles the translation of **case** statements for various distributions of alternatives. The test indicates that at least two different forms of representation are used: a jump table for contiguous ranges of choices and a sequence of comparisons for discontiguous ranges. Tests for other forms of representation produced indeterminate results. This suggests that the compiler optimizes **case** statements using contiguous discrete values and ranges using conceptually efficient strategies. No timing comparison data or size measurement is provided.

- Do special forms of logical expression allowed for **If** statements increase speed?

  The ACEC SSA report includes a test of one short circuit operator for the **If** statement and compares several pairs of **If** statements performing the same test using different encodings of the logical expression (see Observation 3 on page 73).

The **and then** short circuit control form was measured to take somewhat more time than the equivalent nested version (**and then** prevents evaluation of the second term of an expression if the first is false). The measured difference suggests that the **and then** can be used for clarity, but that it does not offer improved performance.

The "Simple Relations" tests (Observation 3 on page 73) test a pair of **If** statements that have equivalent effect using reversed logical expressions. The timings were substantially the same.

- Does the method(s) used to nest the statements affect performance?

  No data available.

- Do alternative test methods, such as table look-up, offer performance advantages?

  The AES SSA compares several alternative logical tests in two comparisons, both titled "Test For Letter Being A Vowel" (Observation 4 on page 75). Using a Boolean array is faster than either the **If** or the **case** statement, while using function calls is slower than the two logical statements.

Configuration(s) tested:

1. **Host**: DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: The tests in this observation examine limits on the **if** and **case** statements.

The AES TG test series examines the compiler for limits that might constrain the user. The stated test objective is to detect unreasonably restrictive characteristics, such as a very small number of enumeration literals, not to find the compiler's maximum capacity. The tests use a relatively coarse binary chop algorithm to determine approximate limits in a reasonable amount of time. When the key phrase "at least" is used, the compiler capacity is greater than the maximum tested value.

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Test TG19, TG32, TG33 and TG36.

```
G.22. TG19

      This test detects whether there is an unreasonably small
      limit to the number of alternatives in a case statement
      permitted by the compiler.

      The number of case statement alternatives was found to be
      at least 5000.
```

G.34. TG32

This test detects whether there is an unreasonably small
limit to the number of statically nested case statements
permitted by the compiler in a compilation unit.

The number of statically nested case statements was found
to be at least 100.

G.35. TG33

This test detects whether there is an unreasonably small
limit to the number of statically nested if statements
permitted by the compiler in a compilation unit.

The number of statically nested if statements was found
to be at least 100.

G.38. TG36

This test detects whether there is an unreasonably small
limit to the number of elsif parts to an if statement
permitted by the compiler.

The number of elsif parts was found to be at least 100.

Observation 2: The tests in this observation check how **case** statements are represented.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0, Test TO17A-F.

0.17. TO17 group of tests

There are six tests in this section. Their purpose is to
examine "case" statements with various distributions of
alternatives to see if the compiler optimises them in the
way one would expect.

Test TO17A examines a "case" statement with an ordered
contiguous range of alternatives. One would expect the
compiler to implement this as a jump table.

The timings produced by this test indicate that the
compiler implements a jump table.

Test TO17B examines a "case" statement with a disordered
contiguous range of alternatives. One would expect the
compiler to implement this as a jump table.

The timings produced by this test indicate that the
compiler implements a jump table.

Test T017C examines a "case" statement with an ordered
contiguous set of ordered contiguous ranges of
alternatives. One would expect the compiler to implement
this as a sequence of comparisons.

The timings produced by this test indicate that the
compiler implements a sequence of comparisons.

Test T017D examines a "case" statement with a sparse
random range of alternatives. One would expect the
compiler to implement this as a binary chop.

Test failed. Aborted by user request
TEST.TST did not set RESULTANT_STATE

☞ Note: Test appeared to run indefinitely.

Test T017E examines a "case" statement with a dense
random range of alternatives. One would expect the
compiler to implement this as a hashed jump table.

The timings produced by this test indicate that the
compiler implements a jump table.

Test T017F examines a "case" statement with few explicit
choices and most of the alternatives in 'others'. One
would expect the compiler to implement this as a sequence
of comparisons.

The timings produced by the test were too inconclusive to
suggest any particular method.

Observation 3: The ACEC compares several pairs of logically equivalent If statements. Each pair
produces the same logical result by different means.

The **and If** short circuit operator is slower than a nested **If** pair. Use of a **not** to reverse a test result is
slower than reversing the sense of the test (presumably the reversed test benefits from requiring one
fewer logical operation).

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Reports "Coding Style Variations":

- "Nested IF vs. AND THEN"
- "Simple Relations" [Tests ss494 and ss495]
- "Simple Relations" [Tests ss497 and ss496]

```
---------------------------------------------------------------------

                       Coding Style Variations

---------------------------------------------------------------------
Nested IF vs AND THEN
---------------------------------------------------------------------
```

```
Test       Execution    Bar                                      Similar
Name       Time         Chart                                    Groups
----------------------------------------------------------------------------
ss490          4.02      ******************************           |
ss491          4.69      ***********************************      |
----------------------------------------------------------------------------
                        Individual Test Descriptions
----------------------------------------------------------------------------
ss490 FOR i IN 1..2
      LOOP
          IF ii = 0
          THEN
              IF bool
              THEN
                  die
              END IF;
          END IF;
          ii := 1 - ii;
      END LOOP;
----------------------------------------------------------------------
ss491 FOR i IN 1..2
      LOOP
          IF ii = 0 AND THEN bool
          THEN
              die;
          END IF;
          ii := 1 - ii;
      END LOOP;
----------------------------------------------------------------------------


----------------------------------------------------------------------------
Simple Relations
----------------------------------------------------------------------------
Test       Execution    Bar                                      Similar
Name       Time         Chart                                    Groups
----------------------------------------------------------------------------
ss494          0.77      *************************                |
ss495          1.08      ***********************************      |
----------------------------------------------------------------------------
                        Individual Test Descriptions
----------------------------------------------------------------------------
----------------------------------------------------------------------------
ss494 IF ll = mm          THEN die; END IF;
----------------------------------------------------------------------------
ss495 IF NOT ( ll /= mm ) THEN die; END IF;
----------------------------------------------------------------------------


----------------------------------------------------------------------------
Simple Relations
----------------------------------------------------------------------------
Test       Execution    Bar                                      Similar
Name       Time         Chart                                    Groups
----------------------------------------------------------------------------
ss497          4.28      **********************************       |
ss496          4.39      **********************************       |
----------------------------------------------------------------------------
                        Individual Test Descriptions
----------------------------------------------------------------------------
----------------------------------------------------------------------------
ss496 IF NOT Bool AND ll = mm  THEN  die;  ELSE  proc0;  END IF;
----------------------------------------------------------------------------
ss497 IF bool OR ll /= mm   THEN  proc0;  ELSE die;  END IF;
----------------------------------------------------------------------------
----------------------------------------------------------------------------
```

Observation 4: The ACEC compares logical tests by implementing the same test with different logical statements. **Case** and **If** statements are compared along with some additional test methods. All the tests perform a logical test to determine if a single character is a vowel.

The **case** statement is faster than the **If** statement, and both are slower than a table lookup.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Version 2.0, SSA Report "Coding Style Variations"

- "Test for Letter Being a Vowel" [Tests ss486, ss488, ss489, ss492 and ss487]
- "Test for Letter Being a Vowel" [Tests ss479, ss482, ss481, ss493 and ss480]

```
------------------------------------------------------------------------

                         Coding Style Variations

------------------------------------------------------------------------
Test For Letter Being A Vowel
------------------------------------------------------------------------
Test       Execution    Bar                                   Similar
Name       Time         Chart                                 Groups
------------------------------------------------------------------------
ss486          1.55     ************                              |
ss488          2.50     *******************                       |
ss489          3.25     *************************                   |
ss492          3.58     ****************************                 |
ss487          4.50     ***********************************            |
------------------------------------------------------------------------
                        Individual Test Descriptions
------------------------------------------------------------------------
ss486 bool := is_a_vowel_1 ( char ) ; -- array
------------------------------------------------------------------------
ss487 bool := is_a_vowel_2 ( char ) ; -- local function
------------------------------------------------------------------------
ss488 CASE char IS
          WHEN 'A'|'E'|'I'|'O'|'U' => bool := True;
          WHEN OTHERS => bool := False;
      end case;
------------------------------------------------------------------------
ss489 bool := char='A' OR char='E' OR char='I' OR char='O' OR char='U';
------------------------------------------------------------------------
ss492 bool := is_a_vowel_3 ( char ); -- function in external package
------------------------------------------------------------------------



------------------------------------------------------------------------
Test For Letter Being A Vowel
------------------------------------------------------------------------
Test       Execution    Bar                                   Similar
Name       Time         Chart                                 Groups
------------------------------------------------------------------------
ss479         80.90     ***************                          |
ss482        110.60     *********************                     |
ss481        126.00     ************************                   |
ss493        178.80     ********************************            |
ss480        186.10     *********************************           |
------------------------------------------------------------------------
                        Individual Test Descriptions
------------------------------------------------------------------------
```

```
ss479 char := 'A';
      WHILE char <= 'Z'
      LOOP
         IF is_a_vowel_1 ( char )  THEN  proc0;  END IF;
                       -- is_a_vowel_1 is boolean array of char
         char := character'succ ( char );
      END LOOP;
----------------------------------------------------------------
ss480 char := 'A';
      WHILE char <= 'Z'
      LOOP
         IF is_a_vowel_2 ( char )  THEN  proc0;  END IF;
              -- is_a_vowel_2 is local function returning boolean
         char := character'succ ( char );
      END LOOP;
----------------------------------------------------------------
ss481 char := 'A';
      WHILE char <= 'Z'
      LOOP
         IF char='A' or char='E' or char='I' or char='O' or char='U'
          THEN      proc0;
          END IF;
          char := character'succ ( char );
      END LOOP;
----------------------------------------------------------------
ss482 char := 'A';
      WHILE char <= 'Z'
      LOOP
         CASE char IS
             when 'A'|'E'|'I'|'O'|'U' => proc0;
             when others=>null;
         END CASE;
         char := character'succ ( char );
       END LOOP;
----------------------------------------------------------------
ss493 char := 'A';
      WHILE char <= 'Z'
      LOOP
         IF is_a_vowel_3 ( char )  THEN  proc0;  END IF;
           -- is_a_vowel_3 is function in external package
         char := character'succ ( char );
       END LOOP;
----------------------------------------------------------------
```

References

- none

2.10 Loop Efficiency

Question: Do different loop constructs vary in efficiency?

Summary: The most efficient loop construct is the for loop (forward or reverse). Loop exit and optimizations are discussed below.

Discussion: Ada provides several statements which can be used to construct loops. These include **for**, **while**, **loop**, and **goto** based loops. Since the loops can be made equivalent, the application programmer can select the appropriate construct based on considerations of style and performance.

Performance of comparable loop constructs should explore raw speed and the variation caused by different forms of indices.

Available tests performing direct comparison of likely loop constructs conclude that the **for** loop (forward and in reverse) is the fastest loop construct. However, unrolled loops are even faster. The available tests do not explore variation in the range specification adequately.

Observation 2 explores some loop optimizations. Optimization necessarily exploits a large number of techniques, which apply to special cases. Of particular note for high performance applications, loops are not unrolled, which suggests that hand-unrolling may be a useful tactic for maximum performance. However, tests on arrays with loop assignment compared to hand-unrolled equivalents suggests that execution speed for hand-unrolled array operations should be checked for individual cases rather than assuming hand-unrolled loops for arrays will reduce execution time (see Arrays on page 7).

Observation 3 shows that the **exit** statement is the most effective means of short-circuiting a loop.

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: Both the AES and ACEC measure the performance of loop constructs and compare their execution speed. Both conclude that the for loop is the fastest loop construct for configuration #1.

The ACEC Single System Analysis (SSA) report compares times for several styles of loops repetitively calling a procedure (included below). The best time value is for an "unrolled" loop (i.e., no loop, a series of procedure calls). The best times for actual loops are for **for** loops in forward and reverse directions. These results are for small loops with constant loop parameters. The design used measures the execution time for the whole execution of the loop for a fixed number of iterations.

The AES test TI11 compares four loop constructs (see below). The **for** loop is the most efficient construct. The test design attempts to measure only the execution time for a single loop iteration, factoring out the time used to initialize and exit the loop. The test loop parameters are based on the clock resolution, rather than having a fixed value. (The loop iteration value varies according to the clock resolution, rather being a value fixed at compilation.)

ACEC Test Results:

Configuration 1, ACEC Release 2.0, Tests TI02A-F.

```
-----------------------------------------------------------------------

                         Coding Style Variations

-----------------------------------------------------------------------
LOOP variations
-----------------------------------------------------------------------
Test        Execution   Bar                                 Similar
Name        Time        Chart                               Groups
-----------------------------------------------------------------------
ss642         22.10     *******************                      |
ss105         27.40     ************************                 |
ss387         27.50     *************************               |
ss385x        30.60     ****************************           |
ss183         30.90     *****************************          |
ss182         32.00     ******************************        |
ss386         36.60     ************************************        |
ss385         36.70     ************************************       |
ss184         37.80     *************************************        |
-----------------------------------------------------------------------
                       Individual Test Descriptions
-----------------------------------------------------------------------
ss105 FOR i in 1..10 LOOP      proc0 ;     END LOOP ;
-----------------------------------------------------------------------
ss182 ii := 10 ;
      LOOP     proc0 ; ii := ii - 1 ;  EXIT WHEN ii <= 0 ;
      END LOOP ;
-----------------------------------------------------------------------
ss183 ii := 10 ;
      LOOP     proc0 ; ii := ii - 1 ;
               IF ii <= 0 THEN  EXIT ;  END IF ;
      END LOOP  ;
-----------------------------------------------------------------------
ss184 ii := 10 ;  proc0 ;  ii := ii - 1 ;
      LOOP    EXIT WHEN ii <= 0 ;  proc0 ;  ii := ii - 1 ;
      END LOOP ;
-----------------------------------------------------------------------
ss385 ii := 10 ;
      WHILE ii > 0 LOOP     proc0 ;  ii := ii - 1 ;
      END LOOP ;
-----------------------------------------------------------------------
ss385x ii := 10 ;
      <<label_for_goto_version>>
         proc0 ;  ii := ii - 1 ;
         IF ii > 0 THEN  GOTO label_for_goto_version ;  END IF ;
-----------------------------------------------------------------------
ss386 ii := 10 ;
      LOOP    EXIT WHEN ii <= 0 ;  proc0 ;  ii := ii - 1 ;
      END LOOP ;
-----------------------------------------------------------------------
ss387 FOR i IN REVERSE 1..10  LOOP     proc0 ;    END LOOP ;
-----------------------------------------------------------------------
```

```
ss642 proc0 ; ... proc0 ;  -- sequence of 10 procedure calls
```

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Tests TI02A-F.

I.26. TI11

This test examines the runtime efficiency of loop
statements. The time per iteration for each of the four
types of loop, using integer loop control variables is
measured.

```
+--------------------+----------------+
|  Type of loop      | Cpu Overhead   |
|                    | per iteration  |
+--------------------+----------------+
| Simple loop        |    1.66us      |
| WHILE loop         |    1.61us      |
| Forward FOR loop   |    409ns       |
| Reversed FOR loop  |    573ns       |
+--------------------+----------------+
```

Observation 2: Both the ACEC and AES test suites provide information on the various loop optimizations (see included text below).

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Report "Optimization:"

- "Loop Unrolling," page 79

- "Loop Flattening : 2 Dimensional Arrays of Real," page 80

- "Loop Invariant Motion" [Tests ss212 and ss3], page 80

- "Loop Invariant Motion" [Tests ss429 and ss430], page 81

- "Loop Invariant Motion" [Tests ss536 and ss535], page 81

- "Loop Invariant Motion" [Tests ss752 and ss11], page 81

- "FOR LOOP with NULL Body," page 82

```
---------------------------------------------------------------------

                          Optimizations

---------------------------------------------------------------------
Loop Unrolling
---------------------------------------------------------------------
Test       Execution    Bar                              Similar
Name       Time         Chart                            Groups
---------------------------------------------------------------------
ss238        0.78       *************                 |
ss3          0.78       *************                 |
ss17         1.41       ***********************         |
ss57         1.42       ***********************         |
```

```
ss240        2.07   ***********************************         |
```

                    Individual Test Descriptions

```
ss3  xx := yy ;
-- Assignment of two floating point variables, library scope.
```

```
ss17 e1 ( ei ) := one ;
-- assignment to one dimensional array of real.
```

```
ss57 ( i ) := one ;   -- i is LOOP index
-- Test subscript computation using FOR LOOP index.
```

```
ss238 FOR i IN 1..1 LOOP    e1 ( i ) := one ;   END LOOP ;
-- can unroll LOOP into single assignment statement
-- simple example amenable to LOOP unrolling
```

```
ss240 FOR i IN 1..2 LOOP    e1 ( i ) := one ;   END LOOP ;
-- simple example amenable to LOOP unrolling
```

    Loop flattening : 2 Dimensional Arrays Of Real

Test Name	Execution Time	Bar Chart	Similar Groups
ss18	3.96	*	|
ss405	156.40	***********************************	|

                    Individual Test Descriptions

If time to execute ss405 is less than 100 times the time to execute
ss18, then the compilation system is treating subscript calculations
using for loop indexes better than general usage. May be using
strength reduction, register allocation, or other techniques including
loop flattening. Flattening is the merging of the two nested loops
into one larger loop.

```
e2 : ARRAY ( int'(1)..int'(10) ,int'(1)..int'(10) ) OF real
            := ( int'(1)..int'(10) =>( int'(1)..int'(10) =>1.0));
ei, ej, ek : int := 1;
```

```
ss18  e2 ( ei, ej ) := one ;
    -- assignment to two dimensional array of real. Checking.
```

```
ss405 FOR i IN 1 .. 10 LOOP
        FOR j IN 1 .. 10 LOOP
            e2 ( int ( i ), int ( j )  ) := one ;
        END LOOP ;
     END LOOP ;
    -- nested FOR loop to access a 2D array -- loops could be flattened
```

    Loop Invariant Motion

        Description   Optimized?

Time : ss212 (9.9) vs ss3 (0.8)   no

```
------------------------------------------------------------------------
ss212 FOR i IN 1..10 LOOP   xx := yy ;   END LOOP ;
-- example where invariant motion is possible
------------------------------------------------------------------------
ss3   xx := yy ;
------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
  Loop Invariant Motion
------------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------------
  Time : ss429 (         3.2 ) vs ss430 (         3.2 )    yes
------------------------------------------------------------------------
```

```
      -------------------- ------------------------------------------
      FUNCTION a1 ( i : int ) RETURN int IS
         ca1 : CONSTANT ARRAY ( int'(0)..int'(2) ) OF int := (0, 1, 2) ;
      BEGIN
         RETURN ca1 ( i ) ;
      END a1 ;
      ------------------------------------------------------------------
      ss429 ii := a1 ( ei ) ;
      -- Is constant static array promoted to outer level?
      ------------------------------------------------------------------
      ca2 : CONSTANT ARRAY ( int'(0)..int'(2) ) OF int := (0, 1, 2) ;
      FUNCTION a2 ( i : int ) RETURN int IS
      BEGIN
         RETURN ca2 ( i ) ;
      END a2 ;
      ------------------------------------------------------------------
      ss430 ii := a2 ( ei ) ; --non-local constant array
      -- Is constant static array promoted to outer level?
      ------------------------------------------------------------------
```

```
------------------------------------------------------------------------
Loop Invariant Motion
------------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------------
  Time : ss536 (       283.1 ) vs ss535 (        91.7 )    nostatistics
------------------------------------------------------------------------
```

```
      ------------------------------------------------------------------
      ss536 FOR l IN 1..mm LOOP
              xx := 0.0 ;
              FOR k IN el'RANGE LOOP
                 xx := xx + el ( k ) ** 2 ;
              END LOOP ;
           END LOOP ;   -- xx is computed from invariants in 'l' loop
      -- very smart optimizer can do inner loop once
      ------------------------------------------------------------------
      ss535 xx := 0.0 ;
           FOR k IN el'RANGE LOOP
              xx := xx + el ( k ) ** 2 ;
           END LOOP ;   -- sample to embed in code for ss536
      ------------------------------------------------------------------
```

```
------------------------------------------------------------------------
Loop Invariant Motion
------------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------------
Time : ss752 (         9.9 ) vs ss11 (        0.8 )    nostatistics
------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------
    ss752 FOR i IN 1..10 LOOP   ii := jj ;   END LOOP ;
    -- could be optimized into an assignment statement, ss11
-------------------------------------------------------------------------
    ss11  kk := ll ;   -- Library scope integer assignment.
-------------------------------------------------------------------------


------------------------- ----------------------------------------------
FOR LOOP with NULL body
-------------------------------------------------------------------------
    Description                                        Optimized?
-------------------------------------------------------------------------
Time : ss106 (          5.8 ) vs ss0 (          0.0 )      nostatistics
-------------------------------------------------------------------------

-------------------------------------------------------------------------
    ss106 FOR i IN 1..10
          LOOP
              NULL ;
          END LOOP ;   -- noop
    -- FOR loop with null body, could be noop.
-------------------------------------------------------------------------
    ss0   NULL ;
-------------------------------------------------------------------------
```

AES Test Results:

Configuration 1, DIY_AES Version 2.0, Tests TI03.

```
0.3.  TOC3

      This test checks loop optimisations.

      Some  loop  invariant  array  addressing  expressions are
      hoisted out of the body of a loop.

      Some loop invariant statements are  hoisted  out  of  the
      body of a loop.

      Simple loops iterating twice are not unrolled.

      Simple loops iterating three times are not unrolled.

      Constraint  checks  are  not  removed  from the body of a
      loop.
```

Observation 3: The ACEC tests the efficiency of various methods for exiting loops. The **exit** statement with logical test was more efficient than either an **if** statement enclosing a simple **exit** or a **goto** statement.

The condition for exit was a simple "greater than" test.

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Report "Language Feature Overhead" - "EXIT from FOR LOOP."

```
------------------------------------------------------------------------
                        Language Feature Overhad

------------------------------------------------------------------------
EXIT From FOR LOOP
------------------------------------------------------------------------
Test        Execution   Bar                                    Similar
Name          Time      Chart                                  Groups
------------------------------------------------------------------------
ss354          3.49     *****************************       |
ss355          4.12     *******-*************************     |
ss356          4.15     *********************************      |
------------------------------------------------------------------------
                     Individual Test Descriptions
------------------------------------------------------------------------
Alternative methods of specifying the same actions.
-----------------------------------------------------------------
ss354 FOR i IN 1..ej
      LOOP
         proc0 ;
         EXIT WHEN l1 > 0 ;
      END LOOP ;
-- EXIT from FOR LOOP with "EXIT WHEN"
-----------------------------------------------------------------
ss355 FOR i IN 1..ej
      LOOP
         proc0 ;
         IF l1 > 0
         THEN
             EXIT ;
         END IF ;
      END LOOP ;
-- EXIT from FOR LOOP with "IF ... THEN EXIT"
-----------------------------------------------------------------
ss356 FOR i IN 1..ej
      LOOP
         proc0 ;
         IF l1 > 0
         THEN
             GOTO q ;
         END IF ;
      END LOOP ;
      <<q>> NULL ;
-- EXIT from FOR LOOP with "IF ... THEN GOTO"
------------------------------------------------------------------------
```

References

- none

2.11 Module Size

Question: Is the performance of a program divided into modules different from a monolithic design?

Summary: Compared to a monolithic design, the execution time of a modular program can be expected to increase to accommodate the costs of calling the procedures used for partitioning. The percentage increase could not be quantified, since it depends upon the application. No information about the size effects of module use is provided. Compiler support for partitioning of a program into modules was measured statically. There are no limitations detected in compiler support.

Discussion: While the division of programs into modules is recommended for program clarity and maintainability, there may be performance consequences. If, for instance, division into smaller modules adds a significant number of procedure calls, the execution time may increase. The use of procedures might reduce program size by allowing more code sharing, or increase it due to additional code inserted for procedure calls.

Issues relating to the use of modules include:

- Is the number of modules supported by the compiler limited?
- How does performance change for programs using smaller modules?
- How is program size affected by use of small modules?

Compiler Limits

When numerous modules are used for a large software program, limits may be found in the compilation system. Some limits relate to the exhaustion of finite physical resources, such as secondary storage or system memory, while other limits might be caused by fixed sizes in the compilation system itself. Detecting such limits reliably is difficult and expensive, since host system configurations are complex combinations of hardware and software.

Observation 1 on page 87 presents the results of limit testing which are relevant to modules. There are no observed limits which strongly preclude the use of modules. These tests do not attempt to overload the library system, and, therefore, do not predict the maximum possible program size.

Performance

Performance change due to the use of modules is conventionally addressed by measuring the overhead of entering and leaving the module (procedure). This cost varies with the complexity of the procedure call and is a combination of overhead relating to the transfer of control and the amount and type of data which are transferred.

Observation 2 on page 88 provides information on overhead, and Observation 3 on page 95 presents some supplemental information on the number of parameters. Some fairly obvious conclusions can be drawn from this data:

- The time required to call a procedure varies with the type of procedures.
- The time required to call a procedure increases as the amount of data transferred increases.

The percentage change in execution time caused by modules cannot be determined solely from measurement of calling overhead. If we assume that the time required to switch to a procedure is independent of the time required to complete the work within a procedure, the time overhead for dividing a program into small modules clearly relates the amount of work done by each module compared to the number of times a module is called. Thus the overhead of procedures can only be calculated when the execution times for procedure bodies is known or reliably estimated and the calling profile for the program is known. This information is not available for this report.

From the measurement of procedure overhead, the performance cost of using modules is minimized by:

- Picking subdivision points to minimize the number of calls. (Inlining should be considered for frequently called smaller procedures.)
- Keeping the number of arguments as small as possible, using constrained types where possible.
- Avoiding trivial procedure bodies: the procedure should perform an appreciable unit of work. This requirement can be relaxed if inlining is used.

The placement of procedures locally or in packages external to the test routine does not appear to have a strong effect on performance.

The performance effects of using common data areas to avoid passing arguments to procedures was not examined. Since Ada procedures are fully reentrant and potentially callable from multiple tasks, this expedient should be carefully considered before use.

Size

Program size can be changed by subdivision. Subdivision adds extra code and storage for movement between modules but reduces program size when code can be shared. The use of inlining and generic procedures also can alter the size of a program.

Size issues were not tested during the preparation of this report.

Generics and Inlining

The behavior of inline and generic routines is addressed in other segments of this report. The speed of inlined procedures approaches that of user-prepared inline code, while generic units impose a small but measurable increase in execution speed. No information was collected on size effects of inlining or generics.

Configuration(s) tested:

1.   **Host**: DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: The AES provides limit testing, including some tests relevant to the use of modules. None of the limit tests attempts to stress the capacity limits of the compiler. No unreasonably small limits were detected.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0, Group G, Tests TG07, TG08, TG24, TG25, TG26, TG28 and TG30.

```
G. Group G - Compiler Capacity Tests

G.8.  TG07

      This test determines whether  there  is  an  unreasonably
      small  limit  to  the  number  of  WITHed  units  in  a
      compilation unit.

      The number of WITHed units was found to be at least 50.


G.9.  TG08

      This test determines whether  there  is  an  unreasonably
      small  limit to the number of USEd units in a compilation
      unit.

      The number of USEd units was found to be at least 10.


G.26. TG24

      This test detects whether there is an unreasonably  small
      limit  to  the  number  of  subprograms  permitted by the
      compiler in a compilation unit.  The  test  is  performed
      with  equal  numbers  of  parameterless  procedures  and
      functions.

      The number of subprograms was found to be between 212 and
      324.


G.27. TG25

      This test determines whether there  is  any  unreasonably
      small  limit  to  the  number of packages permitted by the
      compiler in a compilation unit.
```

The number of packages was found to be at least 50.

G.30. TG28

This test detects whether there is an unreasonably small
limit to the number of statically nested subprograms
permitted by the compiler.

The number of statically nested subprograms was found to
be between 32 and 43.

G.32. TG30

This test detects whether there is an unreasonably small
limit to the number of statically nested subunits
permitted by the compiler in a compilation unit.

The number of statically nested subunits was found to be
between 10 and 21.

Observation 2: Numerous tests of procedure call overhead are provided by the test suites. Their results are provided below.

The location of the procedure (local or external) does not appear to have a large effect on performance. The amount of data transferred as parameters has a large effect on calling speed, while the type of argument has some effects. However, in comparing the type of arguments, size effect, exemplified by the actual number of storage units transferred, should be considered. For instance, the ACEC report "Call Procedures with multiple parameters" (page 91) compares calls using 8 integer and 8 floating point arguments. To compare the tests, definitions of the two types **int** and **real** must be examined (for the Verdix compiler, the definitions could be represented by 4 bytes each).

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Report "Language Feature Overhead":

- "Function versus Procedure," page 88
- "Procedure Calls with Null Body," page 89
- "Subprogram Calls - Formal Generic Procedure," page 90
- "Subprogram Calls - Call Local Nested Procedure," page 90
- "Call Procedures with Multiple Parameters," page 91
- "Subprogram Calls - Parameters : Unconstrained Formal," page 91
- "Subprogram Calls-Parameters : Unconstrained Array Parameter," page 91
- "Subprogram Calls-Parameters : Constrained Record Parameter," page 92

```
----------------------------------------------------------------------

                        Language Feature Overhead

----------------------------------------------------------------------
Function versus Procedure
----------------------------------------------------------------------
Test      Execution     Bar                              Similar
```

```
Name       Time      Chart                                    Groups
-----------------------------------------------------------------------
ss247      9.98      ******************                          |
ss248      20.10     *********************************           |
-----------------------------------------------------------------------
                     Individual Test Descriptions
-----------------------------------------------------------------------
This test compares returning a function value and an out mod
parameter from a procedure.
-----------------------------------------------------------------------

TYPE rec_array IS ARRAY ( int'(1)..int'(4) ) OF byte ;
SUBTYPE c2 IS string ( 1..2 ) ;
subtype c8 is string ( 1..8 ) ;
ccc , hex  : c8;

FUNCTION f ( x : real ) RETURN c8 IS
BEGIN
   IF x > 0.0 THEN RETURN "*********";
              ELSE RETURN "         ";
   END IF ;
END f ;

PROCEDURE p ( x :IN real ; c : OUT c8 ) IS
BEGIN
   IF x > 0.0 THEN c := "*********" ;
              ELSE c := "          " ;
   END IF ;
END p ;
-----------------------------------------------------------------------
ss248  p ( one , ccc ) ;
       -- procedure which returns string
-----------------------------------------------------------------------
ss247  ccc := f ( one ) ;
       -- function which returns string
-----------------------------------------------------------------------



-----------------------------------------------------------------------
Procedure Calls With Null Body
-----------------------------------------------------------------------
Test      Execution   Bar                                    Similar
Name      Time        Chart                                  Groups
-----------------------------------------------------------------------
ss0       0.00                                                  |
ss36      2.30        ******************************            |
ss260     2.51        ********************************          |
-----------------------------------------------------------------------
                     Individual Test Descriptions
-----------------------------------------------------------------------
PROCEDURE n IS
BEGIN
   NULL ;
END n ;
-----------------------------------------------------------------------
ss0    NULL ;
       -- Language feature test, null statement
-----------------------------------------------------------------------
ss36   proc0 ;
       -- call on external null procedure
-----------------------------------------------------------------------
ss260  n ;
       -- local procedure call, body is null
-----------------------------------------------------------------------
```

```
---------------------------------------------------------------------------
     Subprogram Calls - Formal Generic Procedure
---------------------------------------------------------------------------
Test     Execution   Bar                                         Similar
Name     Time        Chart                                       Groups
---------------------------------------------------------------------------
ss36        2.30     ***********************          |
ss478       3.36     ********************************    |
---------------------------------------------------------------------------
                      Individual Test Descriptions
---------------------------------------------------------------------------

---------------------------------------------------------------------------
ss36   proc0 ;
     -- call to library scope procedure with no parameters; body is null.
---------------------------------------------------------------------------
ss478  p0467t04 ;
     -- call procedure which is a generic formal parameter. The actual
     -- procedure is proc0; an external procedure with a null body.
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Subprogram Calls - Call Local Nested Procedure
---------------------------------------------------------------------------
Test     Execution   Bar                                         Similar
Name     Time        Chart                                       Groups
---------------------------------------------------------------------------
ss361       4.32     ********************************    |
ss360       4.69     ********************************    |
---------------------------------------------------------------------------
                      Individual Test Descriptions
---------------------------------------------------------------------------
PROCEDURE L0 IS      -- same body as L2, but at different
BEGIN                -- structural level from it's calling point
  proc0;
END L0;

PROCEDURE L1 IS
   PROCEDURE L2 IS
   BEGIN
      proc0;
   END L2;

   PROCEDURE L3 IS
      PROCEDURE L4 IS
      BEGIN
         PRAGMA include("startime");
            L2;
         PRAGMA include("stoptime0");
         put("ss361 L2; -- null procedure at non-main nesting level");
         PRAGMA include("stoptime2");
      END L4;
   BEGIN    -- body of L3
      L4;
   END L3;
BEGIN        -- body of L1
  L3;
END L1;
---------------------------------------------------------------------------
ss360   L0 ;
      -- call local procedure
---------------------------------------------------------------------------
ss361   L1 ;
      -- call null procedure at non-main nesting level
```

```
----------------------------------------------------------------------


----------------------------------------------------------------------
Call Procedures with multiple parameters
----------------------------------------------------------------------
Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
----------------------------------------------------------------------
ss584           9.67     *********                                !
ss585          38.30     ********************************         !
----------------------------------------------------------------------
                    Individual Test Descriptions
----------------------------------------------------------------------
PROCEDURE pi8 ( i1 , i2 , i3 , i4 , i5 , i6 , i7 , i8 : IN int ) IS
BEGIN
    ii := i1 + i2 + i3 + i4 + i5 + i6 + i7 + i8 ;
END pi8;


PROCEDURE pf8 ( f1 , f2 , f3 , f4 , f5 , f6 , f7 , f8 : IN  real ) IS
BEGIN
    xx := f1 + f2 + f3 + f4 + f5 + f6 + f7 + f8 ;
END pf8;
-----------------------------------------------------------
ss584 pi8 ( 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 ) ;
    -- call procedure with 8 integer parameters
-----------------------------------------------------------
ss585 pf8 ( 1.0 , 2.0 , 3.0 , 4.0 , 5.0 , 6.0 , 7.0 , 8.0 ) ;
    -- call procedure with 8 float parameters
----------------------------------------------------------------------



----------------------------------------------------------------------
Subprogram Calls - Parameters : Unconstrained Formal
----------------------------------------------------------------------
Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
----------------------------------------------------------------------
ss613           2.46     *******************************         !
ss616           2.68     ********************************         !
----------------------------------------------------------------------
                    Individual Test Descriptions
----------------------------------------------------------------------
-----------------------------------------------------------
ss613 proc_record_reference ( pat ) ;
    -- pass parameter to unconstrained formal. Checking enabled.
-----------------------------------------------------------
ss616 proc_record_reference ( pat ) ;
    -- pass parameter to unconstrained formal. Suppress checking.
----------------------------------------------------------------------



----------------------------------------------------------------------
Subprogram Calls-Parameters : Unconstrained Array Parameter
----------------------------------------------------------------------
Test        Execution    Bar                                  Similar
Name        Time         Chart                                Groups
----------------------------------------------------------------------
ss614           2.32     *****************************            !
ss617           2.51     ********************************         !
----------------------------------------------------------------------
                    Individual Test Descriptions
----------------------------------------------------------------------
-----------------------------------------------------------
ss614 proc_vector ( a20 ) ;
```

```
                -- pass parameter to unconstrained formal. Checking enabled.
--------------------------------------------------------------
ss617 proc_vector ( a20 ) ;
                -- pass parameter to unconstrained formal. Suppress checking.
---------------------------------------------------------------------


------------------------------------------------------------------------
Subprogram Calls-Parameters : Constrained Record Parameter
------------------------------------------------------------------------
Test      Execution    Bar                                    Similar
Name      Time         Chart                                  Groups
------------------------------------------------------------------------
ss615        2.22    ********************************    |
ss618        2.35    ********************************    |
------------------------------------------------------------------------
                       Individual Test Descriptions
------------------------------------------------------------------------
------------------------------------------------------------------
ss615 proc_t1_rec ( ra ) ;
                -- pass parameter to unconstrained formal. Checking enabled.
----------------------------------------------------------------
ss618 proc_t1_rec ( ra ) ;
                -- pass parameter to unconstrained formal. Suppress checking.
----------------------------------------------------------------
```

AES Test Results:

Configuration 1, AES Version DIYAES 2.0, Group I, Test TI10.

> Note that most of the inlining tests were removed by optimization (see above). These observations are marked with the phrase ☞**inlining results invalid for the following table**.

Note that the tests of inlined procedures are invalid, with exception of results for inlined integer parameters. The remaining tests of inlined procedure calls record execution times of zero seconds for the inlined routines, which invalidates the comparison to the non-inlined routines. Analysis of the test source code shows that the calling times are measured for paired procedures. The inlined version of the pairs lacks anti-optimization code present in the non-inlined version to access the procedure's argument. Although this anti-optimization code is protected from actual execution during testing, omitting this code allows the compiler to remove all the test code.

```
I. Group I - Runtime Efficiency Tests

I.25. TI10

        This test examines the runtime efficiency of  subprogram
        calls,   in   particular,   the   passing  of  scalar  and
        non-scalar parameters using the "in", "in out", "out" and
        "return" modes. The   test   also   determines   whether   the
        parameters  are  passed  by  copy  or  by  reference. The
        overheads of subprogram entry and exit  are   included   in
        the measured costs. The test is performed for INLINEd and
        non-INLINEd subprograms.

        Test  TO05  indicates that inlining is performed for some
        simple procedures but not for some procedures  which  are
        fairly difficult to inline.
```

Integer parameters:

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	COPY	7.82us	COPY	246ns
in out	COPY	5.90us	COPY	510ns
out	COPY	5.57us	COPY	253ns
return	-	6.04us	-	-

Unconstrained array parameters (array of 256 integers):

☞ **Inlining results invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	7.69us	REF	0s
in out	REF	7.69us	REF	0s
out	REF	8.61us	REF	0s
return	-	229us	-	-

Constrained array parameters (array of 256 integers):

☞ **Inlining results invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	7.23us	REF	0s
in out	REF	6.99us	REF	0s
out	REF	6.79us	REF	0s
return	-	229us	-	-

Simple record parameters (record of 256 components):

☞ **Inlining results invalid for the following table**

Mode	Passing Mechanism	Cpu Time	Passing Mechanism (INLINEd)	Cpu Time (INLINEd)
in	REF	7.37us	REF	0s
in out	REF	6.86us	REF	0s
out	REF	6.79us	REF	0s
return	-	233us	-	-

Discriminated record parameters (record of 256 components):

☞Inlining results invalid for the following table

```
+----------+------------+----------+------------+------------+
|  Mode    |  Passing   |   Cpu    |  Passing   |    Cpu     |
|          | Mechanism  |   Time   | Mechanism  |    Time    |
|          |            |          | (INLINEd)  | (INLINEd)  |
+----------+------------+----------+------------+------------+
|      in  |    REF     |  6.86us  |    REF     |        0s  |
|  in out  |    REF     |  6.84us  |    REF     |        0s  |
|     out  |    REF     |  6.84us  |    REF     |        0s  |
|  return  |     -      |   234us  |     -      |        -   |
+----------+------------+----------+------------+------------+
```

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Tests P000001-P000007.

```
vads ada/optimize=4/errors=(listing,output) 'File_Name'
```

```
Test Name:   P000001                    Class Name:  Procedure
CPU Time:        2.3   microseconds
Wall Time:       2.4   microseconds.     Iteration Count:   2048
Test Description:
 Procedure call and return time ( may be zero if automatic inlining )
 procedure is local
 no parameters
```

```
Test Name:   P000002                    Class Name:  Procedure
CPU Time:        2.4   microseconds
Wall Time:       2.4   microseconds.     Iteration Count:   2048
Test Description:
 Procedure call and return time
 Procedure is local, no parameters
 when procedure is not inlinable
```

```
Test Name:   P000003                    Class Name:  Procedure
CPU Time:        1.9   microseconds
Wall Time:       1.9   microseconds.     Iteration Count:   2048
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 Compare to P000002
```

```
Test Name:   P000004                    Class Name:  Procedure
CPU Time:        0.5   microseconds
Wall Time:       0.5   microseconds.     Iteration Count:   2048
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 pragma INLINE used. Compare to P000001
```

```
Test Name:   P000005                    Class Name:  Procedure
CPU Time:        2.3   microseconds
Wall Time:       2.3   microseconds.     Iteration Count:   2048
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in INTEGER
```

```
Test Name:    P000006                    Class Name:  Procedure
CPU Time:        2.9  microseconds
Wall Time:       2.9  microseconds.       Iteration Count:   2048
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, out INTEGER


Test Name:    P000007                    Class Name:  Procedure
CPU Time:        3.0  microseconds
Wall Time:       3.0  microseconds.       Iteration Count:   2048
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in out INTEGER
```

Observation 3: Tests in this observation examine the effects of the number and ordering of arguments. As the number of arguments increases, the time to access a procedure increases. The ordering of arguments in a procedure call does not appear to have an effect on performance.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Language Feature Overhead":

- "Subprogram Calls-Reference to 1st to 9th Integer Parameter," page 95
- "Subprogram Calls-Reference to 1st to 9th Float Parameter," page 96
- "Procedure with 3 Default Parameters," page 96
- "Subprogram Calls: With 0..3 Parameters," page 97

```
-----------------------------------------------------------------------

                        Language Feature Overhead

-----------------------------------------------------------------------
Subprogram Calls-Reference to 1st to 9th Integer Parameter
-----------------------------------------------------------------------
Test      Execution    Bar                                    Similar
Name      Time         Chart                                  Groups
-----------------------------------------------------------------------
ss566        0.77      ********************************* |
ss567        0.77      ********************************* |
ss568        0.77      ********************************* |
ss571        0.77      ********************************* |
ss574        0.77      ********************************* |
ss569        0.78      *********************************    |
ss570        0.78      *********************************    |
ss572        0.78      *********************************    |
ss573        0.78      *********************************    |
-----------------------------------------------------------------------
                        Individual Test Descriptions
-----------------------------------------------------------------------
-- does system pass first few parameters in registers?
-----------------------------------------------------------------------
ss566   ii := i1 ;      -- reference to first formal integer parameter
-----------------------------------------------------------------------
ss567   ii := i2 ;      -- reference to second formal integer parameter
-----------------------------------------------------------------------
```

```
ss568   ii := i3 ;       -- reference to third formal integer parameter
------------------------------------------------------------------------
ss569   ii := i4 ;       -- reference to fourth formal integer parameter
------------------------------------------------------------------------
ss570   ii := i5 ;       -- reference to fifth formal integer parameter
------------------------------------------------------------------------
ss571   ii := i6 ;       -- reference to sixth formal integer parameter
------------------------------------------------------------------------
ss572   ii := i7 ;       -- reference to seventh formal integer parameter
------------------------------------------------------------------------
ss573   ii := i8 ;       -- reference to eighth formal integer parameter
------------------------------------------------------------------------
ss574   ii := i9 ;       -- reference to ninth formal integer parameter
------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Subprogram Calls-Reference to 1st to 9th Float Parameter
--------------------------------------------------------------------------
Test     Execution   Bar                                        Similar
Name     Time        Chart                                      Groups
--------------------------------------------------------------------------
ss575       0.78     *********************************** |
ss577       0.78     *********************************** |
ss578       0.78     *********************************** |
ss579       0.78     *********************************** |
ss580       0.78     *********************************** |
ss582       0.78     *********************************** |
ss576       0.78     *********************************** |
ss581       0.78     *********************************** |
ss583       0.78     *********************************** |
--------------------------------------------------------------------------
                    Individual Test Descriptions
--------------------------------------------------------------------------
-- does system pass first few parameters in registers?
--------------------------------------------------------------------------
ss575   xx := f1 ;       -- reference to first formal float parameter
------------------------------------------------------------------------
ss576   xx := f2 ;       -- reference to second formal float parameter
------------------------------------------------------------------------
ss577   xx := f3 ;       -- reference to third formal float parameter
------------------------------------------------------------------------
ss578   xx := f4 ;       -- reference to fourth formal float parameter
------------------------------------------------------------------------
ss579   xx := f5 ;       -- reference to fifth formal float parameter
------------------------------------------------------------------------
ss580   xx := f6 ;       -- reference to sixth formal float parameter
------------------------------------------------------------------------
ss581   xx := f7 ;       -- reference to seventh formal float parameter
------------------------------------------------------------------------
ss582   xx := f8 ;       -- reference to eighth formal float parameter
------------------------------------------------------------------------
ss583   xx := f9 ;       -- reference to ninth formal float parameter
------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Procedure With 3 Default Parameters
--------------------------------------------------------------------------
Test     Execution   Bar                                        Similar
Name     Time        Chart                                      Groups
--------------------------------------------------------------------------
ss127       5.52     ****************               |
ss124      11.10     ***********************************   |
ss126      11.20     ***********************************      |
```

```
ss125       11.50     ****************************        |
--------------------------------------------------------------------
                    Individu.l Test Descriptions
--------------------------------------------------------------------
PROCEDURE dry_martini
      ( base   : spirit   := gin ;
        how    : style    := on_the_rocks ;
        with_a : trimming := olive ) IS
BEGIN
    martini ( base, how, with_a ) ;
END dry_martini ;
--------------------------------------------------------------
ss124  dry_martini ; --  Call local procedure with 3 default parameters,
                     --  omitting all parameters on call.
--------------------------------------------------------------
ss125  dry_martini ( gin , on_the_rocks , olive ) ;
                     -- Call local procedure with 3 default parameters,
                     -- specify all parameters on call.
--------------------------------------------------------------
ss126  dry_martini ( how => straight_up ) ;
                     -- Call local procedure with 3 default parameters,
                     -- specify second parameter (by name) on call.
--------------------------------------------------------------
ss127  martini ( gin , on_the_rocks , olive ) ;
                     -- Lower level procedure that ss124-ss126 call on.
--------------------------------------------------------------


--------------------------------------------------------------------
Subprogram Calls: With 0..3 Parameters
--------------------------------------------------------------------
Test      Execution   Bar                                 Similar
Name      Time        Chart                               Groups
--------------------------------------------------------------------
ss36          2.30    ******                                |
ss37          5.97    ****************                       |
ss38          9.39    *************************               |
ss39         13.00    *********************************       |
--------------------------------------------------------------------
                    Individual Test Descriptions
--------------------------------------------------------------------
--------------------------------------------------------------
ss36   proc0 ;
   -- simple procedure with no parameters; call to library scope
   -- procedure : body is null.
--------------------------------------------------------------
ss37   proc1 ( xx ) ;
   -- simple procedure with one IN OUT floating point parameter,
   -- declared in external library unit : body is null.
--------------------------------------------------------------
ss38   proc2 ( xx , yy ) ;
   -- simple procedure with two IN OUT floating point parameters,
   -- declared in external library unit : body is null.
--------------------------------------------------------------
ss39   proc3 ( xx , yy , zz ) ;
   -- simple procedure with three IN OUT floating point parameters,
   -- declared in external library unit : body is null.
--------------------------------------------------------------
```

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Test P000005, P000010-P000013.

```
vads ada/optimize=4/errors=(listing,output) 'File_Name'
```

Test Name: P000005   Class Name: Procedure
CPU Time: 2.3   microseconds
Wall Time: 2.3   microseconds.   Iteration Count: 2048
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in INTEGER

Test Name: P000010   Class Name: Procedure
CPU Time: 8.6   microseconds
Wall Time: 8.6   microseconds.   Iteration Count: 1024
Test Description:
 Procedure call and return time measurement
 Compare to P000005
 10 parameters, in INTEGER

Test Name: P000011   Class Name: Procedure
CPU Time: 18.0   microseconds
Wall Time: 18.0   microseconds.   Iteration Count: 512
Test Description:
 Procedure call and return time measurement
 Compare to P000005, P000010
 20 parameters, in INTEGER

Test Name: P000012   Class Name: Procedure
CPU Time: 11.3   microseconds
Wall Time: 11.3   microseconds.   Iteration Count: 512
Test Description:
 Procedure call and return time measurement
 Compare with P000010 (discrete vs composite parameters)
 10 paramaters, in MY_RECORD a three component record

Test Name: P000013   Class Name: Procedure
CPU Time: 22.1   microseconds
Wall Time: 22.1   microseconds.   Iteration Count: 512
Test Description:
 Procedure call and return time measurement
 twenty composite 'in' parameters
 the composite type is a three component record

References

- none

2.12 Optimization Options

Question: What are the effects of different optimization levels?

Summary: The VADS compiler provides 10 optimization levels, from 0 to 9. The default value is 4. For the tested cases, higher levels of optimization provide only limited improvements in execution performance, at the cost of increased compilation time. Optimization may be deactivated selectively for parts of programs.

Discussion: The Ada language standard provides program control of compiler optimizations by **pragma** OPTIMIZE. The standard options are OPTIMIZE(SPACE) and OPTIMIZE(TIME). A compiler may honor the pragma or not, and can provide additional optimization options.

In deciding how to set optimization levels, the programmer should consider the following questions:

- How do optimization levels affect execution speed?
- How do optimization levels change program size?
- Does optimization cause any undesirable side effects, such as moving variables that require a fixed location in memory?
- What is the cost (or overhead) associated with various optimization levels?

The tested compiler does not honor **pragma** OPTIMIZE. The compiler provides 10 levels of optimization, applied by a compiler command line switch, which applies increasing numbers of optimizations and performs additional passes through the code (Observation 1 on page 100).

Since the optimization of individual code segments depends heavily on that code's exact syntax, it is impossible to make a general statement about the increase in of performance for Ada programs. However, Observation 2 on page 102 shows the effects of increasing execution time on the PIWG B tests, a set of application programs. This illustrates that the improvement in execution speed trails off above optimization level 3.

No information on program size is provided as part of this report.

While both the AES and ACEC provide extensive tests to demonstrate which optimizations are performed, there is no correlation between the use of specific optimizations and potentially harmful side effects due to optimization. As can be seen in Figure 2-4 on page 103, the PIWG Tests B000001A and B failed when the optimization level was set to 4 or higher. The reason for this failure is not known. In another example, programs that control peripherals through successive writes of constant values to specified memory locations may give the appearance code, which an optimizer can remove from the program. Observation 3 on page 104 provides a listing of tests for specific optimizations. The Verdix documentation indicates that optimization can be deactivated selectively for packages, subprograms, and objects (Observation 1 on page 100).

No information on compiler timing or size for the compiler was collected for this report.

Configuration(s) tested:

1.   **Host**: DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and
   25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-
   state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data
   cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb
   DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30,
   Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: Verdix documentation describes available optimizations. Note that **pragma** OPTIMIZE
is not effective, but the level of optimization can be set by a command line option, and that optimization
can be selectively disabled for individual subprograms, packages, and named objects.

Several portions of the Verdix documentation are relevant:

VADS Programmer's Guide, PG F-4:

   pragma OPTIMIZE
      is recognized by the implementation, but has no effect in the cur-
      rent release...

   pragma OPTIMIZE_CODE(OFF|ON)
      specifies whether the code should be optimized (ON) by the com-
      piler or not (OFF). It can be used in any subprogram. When OFF
      is specified, the compiler generates unoptimized code. The default
      is OFF.

      Optimization can be selectively suppressed using this pragma at
      the subprogram level. Inline subprograms are optimized even if
      they have **pragma** OPTIMZE_CODE(OFF) unless the caller also
      has **pragma** OPTIMIZE_CODE(OFF).

VADS Programmer's Guide, PG F-7:

   pragma VOLATILE(*object*)
      guarantees that loads and stores to the named object will be per-
      formed as *expected after optimization.*

VADS User's Guide, UG 4-4:

> The Verdix optimizer performs most classical code optimizations and several that are specific to Ada:
>
> - Code straightening
> - Constant folding, copy propagation and strength reduction
> - Redundant branch and range check elimination
> - Common subexpression elimination including elimination of common expressions from the alternatives of a branch or case statement
> - Hoistings of loop invariant computations and range checks
> - Strength reduction of index computations within a loop
> - Loop induction variables for array indexing within a loop
> - Range propagation for elimination of constraint checking
> - Limitation of assignment to unused local variables
> - Address simplification
>
> In addition, the following VADS compiler features relate to the runtime performance of the generated code:
>
> - Local scalar and access variables automatically allocated in registers
> - Loop variables allocated in registers
> - Parameters passed in registers
> - Graph coloring register allocation scheme
> - Code generation for math coprocessors
> - Target specific peephole optimization
>
> **pragma** OPTIMIZE_CODE(ON|OFF) can suppress or re-enable optimization for a specific subprogram or package.
>
> **pragma** VOLATILE(*object_name*) guarantees that references to the named object will not be optimized away.

VADS User Guide, UG 8-22:

> /OPTIMIZE[=number]Invoke the code optimizer (OPTIM3). An optional digit provides the level of optimization. /OPTIMIZE=4 is the default.
>
> | | |
> |---|---|
> | /OPTIMIZE | no digit, full optimization |
> | /OPTIMIZE=0 | prevents optimization |
> | /OPTIMIZE=1 | no hoisting |
> | /OPTIMIZE=2 | no hoisting but more passes |
> | /OPTIMIZE=3 | no hoisting but even more passes |
> | /OPTIMIZE=4 | hoisting from loops |

/OPTIMIZE=5	hoisting from loops but more passes
/OPTIMIZE=6	hoisting from loops with maximum passes
/OPTIMIZE=7	hoisting from loops and branches
/OPTIMIZE=8	hoisting from loops and branches, more passes
/OPTIMIZE=9	hoisting from loops and branches, maximum passes

Hoisting from branches (and cases alternatives) can be slow and does not always provide significant performance gains so it can be suppressed.

Observation 2: The PIWG "B" tests were run for each optimization level provided by the Verdix compiler. The results are graphed below.
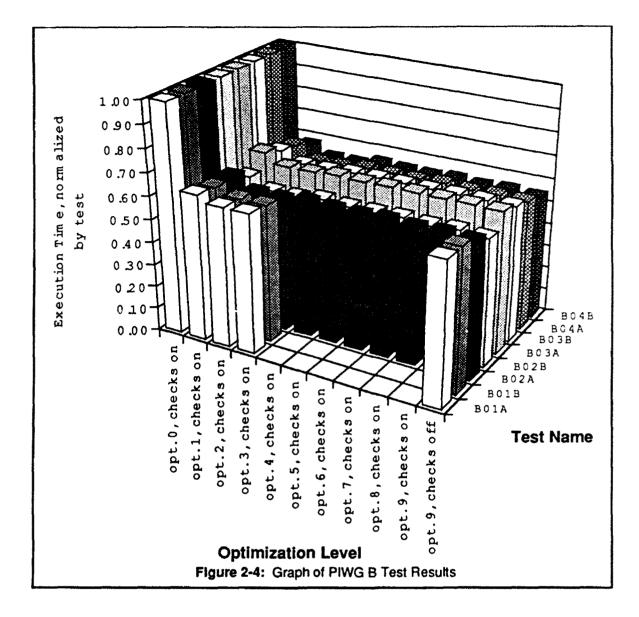
PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Tests B000001-B000004, for optimization level 0-9 (see also Observation 1).

Table 2: Execution Times (in Seconds) for PIWG B Tests at Various Optimization Levels

PIWG Test	Opt. 0[a] Ch[b]	Opt. 1 Ch	Opt. 2 Ch	Opt. 3 Ch	Opt. 4 Ch	Opt. 5 Ch	Opt. 6 Ch	Opt. 7 Ch	Opt. 8 Ch	Opt. 9 Ch	Opt. 9 No[c]
B000001A	46.9	30.0	28.7	28.7							29.2
B000001B	36.9	23.2	22.3	22.2							22.9
B000002A	42.3	26.7	24.7	24.7	25.2	25.0	25.0	25.3	25.0	25.2	25.0
B000002B	35.5	21.0	19.5	19.5	19.9	19.7	19.7	19.9	19.7	19.9	19.7
B000003A	48.1	31.7	29.6	29.5	29.9	29.9	29.9	29.9	29.9	29.9	29.9
B000003B	41.1	25.7	24.1	24.0	24.4	24.5	24.5	24.5	24.5	24.4	24.4
B000004A	46.3	28.2	25.9	25.9	26.6	26.4	26.4	26.6	26.4	26.6	26.4
B000004B	35.7	20.4	19.0	19.0	19.6	19.4	19.4	19.6	19.4	19.6	19.4

a   Opt.=optimization level selected for the Verdix compiler.
b   Ch=run-time checking enabled.
c   No=no run-time checking.

Figure 2-4: Graph of PIWG B Test Results

Notes:

- The Y axis displays the optimization levels provided by the Verdix compiler, from 0 (no optimization) to 9 (full optimization). From left to right, the first 10 observations were made with run-time checking on, while the final observation has checking turned off.

- The X axis shows the PIWG B test name (abbreviated). Each test was attempted 11 times, for various optimization levels. Note the B01A and B01B tests failed for some optimization levels.

- The Z axis shows the execution time, normalized by test. For each test, the execution time for optimization level 0 was taken as the value 1.0, with the other results in the series shown in proportion. This displays all test series to the same scale, showing how optimization affected execution times for the PIWG B tests. *If actual execution times are needed, consult the table above.*

- PIWG Test B000001 failed when the optimization level was set to 4 or above.

Observation 3: Numerous tests for individual optimizations are made by both the ACEC and AES test suites.

<u>AES Test Results:</u>

Observation 1, DIY_AES Version 2.0, Group O, Tests TO01-TO04, TO08, TO10-TO14, TO16, TO18-TO20.

Note that tests appearing in other portions of this report are not reproduced here. See, for example, array efficiency, check suppression and generic procedures.

```
O. Group O - Optimization Tests

O.1.  TO01

      This   test   checks   optimizations   involving   value
      propagation.

      Value propagation is the replacement, at compile time, of
      a reference to a variable by its known  value.  The  code
      size  of  sections  of  code  which  permitted  value
      propagation was compared with an equivalent section where
      propagation could not be performed. The following results
      were obtained.

      Simple  propagation  optimization  is  performed  for
      integers.

      Propagation  of  values  through if statements does occur
      for integers.

      Chains of propagation of  values  (i.e.  remembering  the
      contents  of  variables  as  a  result of assigning other
      variables with known  values  to  them)  does  occur  for
      integers.

      Simple propagation optimization is performed for floats.

      Propagation  of  values  through if statements does occur
      for floats.

      Chains of propagation of  values  (i.e.  remembering  the
      contents  of  variables  as  a  result of assigning other
      variables with known  values  to  them)  does  occur  for
      floats.


O.2.  TO02

      This test checks for common subexpression elimination, in
      particular,  examining  whether common subexpressions are
      only evaluated once.

      Two common  subexpressions  involving  integers  are  not
      recognised as being common.

      Three  common  subexpressions  involving integers are not
      recognised as being common.

      Two common subexpressions involving one-dimensional array
      addressing for integers are recognised  as  being  common
      and only evaluated once.
```

Two common subexpressions involving two-dimensional array
addressing for integers are recognised as being common
and only evaluated once.

Two common subexpressions involving floats are not
recognised as being common.

Three common subexpressions involving floats are
recognised as being common and only evaluated once.

Two common subexpressions involving one-dimensional array
addressing for floats are recognised as being common and
only performed once.

Three common subexpressions involving two-dimensional
array addressing for floats are recognised as being
common and only performed once.

0.3. TO03

This test checks loop optimizations.

Some loop invariant array addressing expressions are
hoisted out of the body of a loop.

Some loop invariant statements are hoisted out of the
body of a loop.

Simple loops iterating twice are not unrolled.

Simple loops iterating three times are not unrolled.

Constraint checks are not removed from the body of a
loop.

0.4. TO04

This test examines the use of registers for variables.

The test was performed by comparing code sizes of groups
of statements which allowed scope for some (but not all)
variables to be allocated to registers. If registers were
used, some of the statements would yield shorter code
sequences. It was found that approximately 4 registers
were used to hold variables.

0.8. TO08

This test determines whether or not only those
subprograms which are referenced are loaded.

The test measures the size of code loaded, to see if
unreferenced subprograms are in fact loaded. Pragma
OPTIMIZE (SPACE) was set and it was noted that
unreferenced subprograms are not loaded.

0.10. TO10

This test checks a variety of subexpression evaluation
optimizations, in particular constant folding.

Constant folding (performing compile-time arithmetic on constants) and special-case expression evaluation (e.g. multiplying by zero or one) was tested under a variety of conditions. Code sizes obtained were compared with control versions which inhibited the optimizations. Pragma OPTIMIZE (SPACE) was used.

Simple constant folding of adjacent constants is performed for integers.

Constant folding after re-arrangement (i.e. the compiler rearranges the expression to bring two constants together) is not performed for integers.

Complex constant folding (e.g. evaluating bracketed compile-time known expressions) is not performed for integers.

Simple constant folding, involving propagation of remembered values from elsewhere in the code, is not performed for integers.

Complex constant folding, involving propagation of remembered values from elsewhere in the code, is performed for integers.

Special-case expression evaluation is performed for integers.

Simple constant folding of adjacent constants is performed for floats.

Constant folding after re-arrangement (i.e. the compiler rearranges the expression to bring two constants together) is not performed for floats.

Complex constant folding (e.g. evaluating bracketed compile-time known expressions) is not performed for floats.

Simple constant folding, involving propagation of remembered values from elsewhere in the code, is not performed for floats.

Complex constant folding, involving propagation of remembered values from elsewhere in the code, is not performed for floats.

Special-case expression evaluation is performed for floats.

O.11. TO11

This test performs checks on the suppression of redundant runtime checks.

Timing tests were run to compare code where runtime checks are required with similar code where the checks are redundant. Pragma OPTIMIZE (TIME) was set.

The index check when indexing arrays is automatically eliminated when not required when the index subtype is a subtype of the array range.

Constraint checks on indexing arrays are not eliminated
when the index value is known by propagation of the value
of the index.

The range check when assigning objects of different
subtypes is automatically eliminated when not required
because one object is a subtype of the object being
assigned to.

O.12. TO12

This test examines the effectiveness of register
allocation and involves expressions requiring storage of
intermediate results.

Test failed. No TEST.TST

O.13. TO13

This test checks whether or not only referenced subunits
are loaded. The test measures the size of code loaded to
see if unreferenced subunits are in fact loaded. Pragma
OPTIMIZE (SPACE) was set.

Unreferenced subunits are not loaded.

O.14. TO14

This test determines whether the compiler can recognise
and remove unreachable and redundant code.

The compiler did not generate unreachable code in a
function where extra statements followed the 'return'
statement.

The compiler did generate redundant code in the case of
an 'if' statement where the result of the condition was
known at compile-time.

O.16. TO16

This test determines whether the compiler can recognise
code which will definitely cause a predefined exception
and replace it with exception raising code.

The compiler did not recognise and replace with exception
raising code, statements which would definitely cause a
NUMERIC_ERROR.

The compiler did recognise and replace with exception
raising code, statements which would definitely cause a
CONSTRAINT_ERROR.

O.18. TO18

The aim of this test is to determine if the compiler can
recognise an accept statement with a null body and avoid
context switching in this case, thus making a significant
time saving.

```
Test failed. No TEST.TST
```

0.19. TO19

The aim of this test is to help determine if there are any special optimizations when the compiler recognises a passive task, ie. turning the task into a monitor package. In this case the passive task simply protects a shared variable.

Test failed. Malfunction in Test Harness
Exception in Unattended mode

0.20. TO20

The aim of this test is to help determine if there are any special optimizations when the compiler recognises a passive task, ie. turning the task into a monitor package. In this case the passive task handles a buffer similar to a MASCOT channel.

Test failed. Malfunction in Test Harness
Exception in Unattended mode

ACEC Test Results:

Observation 1, ACEC Release 2.0, SSA Report "Optimizations":

- "Algebraic Simplification : Array of Integer" [Tests ss432 and ss433], ige 112
- "Algebraic Simplification : Array of Integer" [Tests ss434 and ss435], page 113
- "Algebraic Simplification : Array of Integer" [Tests ss436 and ss437], page 113
- "Algebraic Simplification : Boolean" [Tests ss83 and ss82], page 113
- "Algebraic Simplification : Boolean" [Tests ss85 and ss86], page 113
- "Algebraic Simplification : Boolean" [Tests ss319 and ss320], page 114
- "Algebraic Simplification : Boolean" [Tests ss321 and ss322], page 114
- "Algebraic Simplification : Integer" [Tests ss51 and ss11], page 114
- "Algebraic Simplification : Integer" [Tests ss44 and ss0], page 114
- "Algebraic Simplification : Integer" [Tests ss52 and ss11], page 114
- "Algebraic Simplification : Integer" [Tests ss47 and ss11], page 115
- "Algebraic Simplification : Integer" [Tests ss560 and ss561], page 115
- "Algebraic Simplification : Integer" [Tests ss48 and ss11], page 115
- "Algebraic Simplification : Integer" [Tests ss49 and ss11], page 115
- "Algebraic Simplification : Integer" [Tests ss50 and ss7], page 115
- "Algebraic Simplification : Integer" [Tests ss9 and ss43], page 116
- "Algebraic Simplification : Floating Point" [Tests ss64 and ss3], page 116
- "Algebraic Simplification : Floating Point" [Tests ss61 and ss3], page 116

- "Folding in Inline Function" [Tests ss563 and ss7], page 137
- "Machine Idioms" [Tests ss29 and ss3], page 137
- "Machine Idioms" [Tests ss30 and ss11], page 137
- "Machine Idioms" [Tests ss40 and ss11], page 137
- "Machine Idioms" [Tests ss45 and ss7], page 138
- "Machine Idioms" [Tests ss52 and ss9], page 138
- "Machine Idioms" [Tests ss59 and ss3], page 138
- "Machine Idioms" [Tests ss115 and ss114], page 138
- "Machine Idioms" [Tests ss128 and ss129], page 138
- "Machine Idioms" [Tests ss196 and ss201], page 139
- "Machine Idioms" [Tests ss198 and ss201], page 139
- "Machine Idioms" [Tests ss201 and ss202], page 139
- "Machine Idioms" [Tests ss197 and ss203], page 139
- "Machine Idioms" [Tests ss199 and ss200], page 140
- "Machine Idioms" [Tests ss204 and ss200], page 140
- "Machine Idioms" [Tests ss207 and ss208], page 140
- "Machine Idioms" [Tests ss323 and ss324], page 141
- "Machine Idioms" [Tests ss215 and ss11], page 141
- "Machine Idioms" [Tests ss503 and ss0], page 141
- "Machine Idioms" [Tests ss205 and ss206], page 141
- "Jump Tracing" [Tests ss250 and ss0], page 142
- "Jump Tracing" [Tests ss619 and ss0], page 142
- "Jump Tracing" [Tests ss620 and ss0], page 142
- "Jump Tracing" [Tests ss26 and ss0], page 142
- "Jump Tracing" [Tests ss261 and ss0], page 143
- "Loop Fusion," page 143
- "Loop Interchange", page 143
- "Loop Unrolling, Test Elimination" [Tests ss541 and ss542x], page 144
- "Loop Unrolling, Test Elimination" [Tests ss542 and ss542x], page 144
- "Loop Unrolling" [Tests ss105 and ss642], page 145
- "Loop Unrolling" [ss3, ss17, ss57, ss238, and ss240], page 145
- "Loop Flattening: 2 Dimensional Arrays Of Real," page 145
- "Loop Invariant Motion" [Tests ss212 and ss3], page 146
- "Loop Invariant Motion" [Tests ss429 and ss430], page 146
- "Loop Invariant Motion" [Tests ss536 and ss535], page 146
- "Loop Invariant Motion" [Tests ss752 and ss11], page 147
- "FOR LOOP with NULL Body," page 147

```
--------------------------------------------------------------------

                            Optimizations

--------------------------------------------------------------------
Algebraic Simplification : Array of Integer
--------------------------------------------------------------------
        Description                           Optimized?
--------------------------------------------------------------------
Time : ss432 (        20.9 ) vs ss433 (        18.2 )    nostatistics
```

```
--------------------------------------------------------------------------
                 b := el(1);
                 c := el(2);
                 d := el(3);
                 e := el(4);
                 f := el(5);
--------------------------------------------------------------------------

      --------------------------------------------------------------------
      ss432 => a := b * ( e + f ) - c * ( e + f ) + d * ( e + f ) ); +
      --------------------------------------------------------------------
      ss433 => a := ( b - c + d ) * ( e + f ); +
   -----------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Array of Integer
--------------------------------------------------------------------------
      Description                                Optimized?
--------------------------------------------------------------------------
Time : ss434 (       13.8 ) vs ss435 (        22.4 )      yes
--------------------------------------------------------------------------

      --------------------------------------------------------------------
      ss434 => a := b + c + d * e; +
      --------------------------------------------------------------------
      ss435 => t1 := b + c;   t2 := d * e;   a := t1 + t2; +
   -----------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Array of Integer
--------------------------------------------------------------------------
      Description                                Optimized?
--------------------------------------------------------------------------
Time : ss436 (       13.9 ) vs ss437 (        15.4 )      yes
--------------------------------------------------------------------------

      --------------------------------------------------------------------
      ss436 => a := b / c / d / e; +
      --------------------------------------------------------------------
      ss437 => a := b / ( c * d * e ); +
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Boolean
--------------------------------------------------------------------------
      Description                                Optimized?
--------------------------------------------------------------------------
Time : ss83 (        1.8 ) vs ss82 (        1.4 )      nostatistics
--------------------------------------------------------------------------

      --------------------------------------------------------------------
      ss83  => IF NOT ( ll >= mm ) THEN ii := 1;  END IF; -- True,
      --------------------------------------------------------------------
      ss82  => IF ll < mm THEN ii := 1;  END IF;         -- True,
   -----------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Boolean
--------------------------------------------------------------------------
      Description                                Optimized?
--------------------------------------------------------------------------
Time : ss85 (        2.1 ) vs ss86 (        1.7 )      nostatistics
--------------------------------------------------------------------------

      --------------------------------------------------------------------
      ss85 => IF ll < mm THEN ii := 1;  ELSE die;  END IF;   -- True;
      --------------------------------------------------------------------
      ss86 => IF ll >= mm THEN die; ELSE ii := 0;  END IF;  -- False
   -----------------------------------------------------------------------
```

```
--------------------------------------------------------------------
Algebraic Simplification : Boolean
--------------------------------------------------------------------
      Description                              Optimized?
--------------------------------------------------------------------
Time : ss319 (        1.8 ) vs ss320 (        1.8 )    yes
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss319 =>  IF mm > 11 OR False     THEN ii := 1;  END IF;
      --------------------------------------------------------------
      ss320 =>  IF mm > 11 OR ELSE False THEN ii := 1;  END IF;
   -----------------------------------------------------------------


--------------------------------------------------------------------
Algebraic Simplification : Boolean
--------------------------------------------------------------------
      Description                              Optimized?
--------------------------------------------------------------------
Time : ss321 (        1.9 ) vs ss322 (        0.8 )    nostatistics
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss321 =>  IF mm > 11 OR True     THEN ii := 1;  END IF;
      --------------------------------------------------------------
      ss322 =>  IF mm > 11 OR ELSE True THEN ii := 1;  END IF;
--------------------------------------------------------------------


--------------------------------------------------------------------
Algebraic Simplification : Integer
--------------------------------------------------------------------
      Description                              Optimized?
--------------------------------------------------------------------
Time : ss51 (       0.8 ) vs ss11 (        0.8 )    yes
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss51  => ii := 11 ** 1;
      --------------------------------------------------------------
      ss11  => kk := 11;
   -----------------------------------------------------------------


--------------------------------------------------------------------
Algebraic Simplification : Integer
--------------------------------------------------------------------
      Description                              Optimized?
--------------------------------------------------------------------
Time : ss44 (       0.0 ) vs ss0 (        0.0 )    yes
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss44  => ii := ii + 0;
      --------------------------------------------------------------
      ss0   => NULL;
   -----------------------------------------------------------------


--------------------------------------------------------------------
Algebraic Simplification : Integer
--------------------------------------------------------------------
      Description                              Optimized?
--------------------------------------------------------------------
Time : ss52 (       0.9 ) vs ss11 (        0.8 )    nostatistics
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss52  => ii := 11 + 1;
      --------------------------------------------------------------
      ss11  => kk := 11;
   -----------------------------------------------------------------
```

```
------------------------------------------------------------------------
Algebraic Simplification : Integer
------------------------------------------------------------------------
      Description                               Optimized?
------------------------------------------------------------------------
Time : ss47 (         0.8 ) vs ss11 (         0.8 )    yes
------------------------------------------------------------------------

      -------------------------------------------------------------
      ss47  => ii := ll * 1;
      -------------------------------------------------------------
      ss11  => kk := ll;
   ----------------------------------------------------------------


------------------------------------------------------------------------
Algebraic Simplification : Integer
------------------------------------------------------------------------
      Description                               Optimized?
------------------------------------------------------------------------
Time : ss560 (        0.8 ) vs ss561 (        0.8 )    yes
------------------------------------------------------------------------

      -------------------------------------------------------------
      ss560 =>  ii := -1 * ii;
      -------------------------------------------------------------
      ss561 =>  ii := -ii;
      ----------------------------------------------------------


------------------------------------------------------------------------
Algebraic Simplification : Integer
------------------------------------------------------------------------
      Description                               Optimized?
------------------------------------------------------------------------
Time : ss48 (         0.8 ) vs ss11 (         0.8 )    yes
------------------------------------------------------------------------

      -------------------------------------------------------------
      ss48  => ii := ll / 1;
      -------------------------------------------------------------
      ss11  => kk := ll;
      ----------------------------------------------------------


------------------------------------------------------------------------
Algebraic Simplification : Integer
------------------------------------------------------------------------
      Description                               Optimized?
------------------------------------------------------------------------
Time : ss49 (         0.8 ) vs ss11 (         0.8 )    yes
------------------------------------------------------------------------

      -------------------------------------------------------------
      ss49  => ii := ll * 0;
      -------------------------------------------------------------
      ss11  => kk := ll;
      ----------------------------------------------------------


------------------------------------------------------------------------
Algebraic Simplification : Integer
------------------------------------------------------------------------
      Description                               Optimized?
------------------------------------------------------------------------
Time : ss50 (         0.8 ) vs ss7 (         0.8 )     nostatistics
------------------------------------------------------------------------

      -------------------------------------------------------------
      ss50  => ii := ll ** 0;
      -------------------------------------------------------------
      ss7   => kk := 1;
      -------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Algebraic Simplification : Integer
--------------------------------------------------------------------------
       Description                              Optimized?
---------------------  ----------------------------------------------------
Time : ss9 (           1.1 ) vs ss43 (         3.7 )       yes
--------------------------------------------------------------------------

       ss9  => kk := ll + mm;
       -------------------------------------------------------------------
       ss43 -> ii := 0;   proc0;   ii := ii + 1;
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Floating Point
--------------------------------------------------------------------------
       Description                              Optimized?
--------------------------------------------------------------------------
Time : ss64 (          0.8 ) vs ss3 (          0.8 )       nostatistics
--------------------------------------------------------------------------

       ss64  => xx := yy + 0.0;
       -------------------------------------------------------------------
       ss3   => xx := yy ;
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Floating Point
--------------------------------------------------------------------------
       Description                              Optimized?
--------------------------------------------------------------------------
Time : ss61 (          0.8 ) vs ss3 (          0.8 )       maybe
--------------------------------------------------------------------------

       ss61  => xx := yy * 1.0;
       -------------------------------------------------------------------
       ss3   => xx := yy ;
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Floating Point
--------------------------------------------------------------------------
       Description                              Optimized?
--------------------------------------------------------------------------
Time : ss62 (          0.8 ) vs ss3 (          0.8 )       no
--------------------------------------------------------------------------

       ss62  => xx := yy / 1.0;
       -------------------------------------------------------------------
       ss3   => xx := yy ;
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Floating Point
--------------------------------------------------------------------------
       Description                              Optimized?
--------------------------------------------------------------------------
Time : ss63 (          0.8 ) vs ss3 (          0.8 )       nostatistics
--------------------------------------------------------------------------

       ss63  => xx := yy * 0.0;
       -------------------------------------------------------------------
       ss3   => xx := yy ;
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Algebraic Simplification : Floating Point
--------------------------------------------------------------------------
        Description                                  Optimized?
--------------------------------------------------------------------------
Time : ss65 (          0.8 ) vs ss1 (          0.8 )     nostatistics
--------------------------------------------------------------------------

        --------------------------------------------------------------
        ss65  => xx := yy ** 0;
        --------------------------------------------------------------
        ss1   => xx := 1.0;
        --------------------------------------------------------------


--------------------------------------------------------------------------
Algebraic Simplification : Boolean NOT, NOT NOT
--------------------------------------------------------------------------
    Test        Execut on   Bar                              Similar
    Name        Tim :       Chart                            Groups
--------------------------------------------------------------------------
ss72            0.89    ***************************        |
ss73            1.17    *********************************  |
--------------------------------------------------------------------------
                        Individual Test Descriptions
--------------------------------------------------------------------------

--------------------------------------------------------------------------
ss72 bool := NOT bool ;
-- boolean operator NOT.
--------------------------------------------------------------------------
ss73 bool := NOT ( NOT bool ) ; -- could be noop
-- algebraic simplification; boolean NOT NOT.
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Range Constraint Check
--------------------------------------------------------------------------
Test        Execution   Bar                                  Similar
Name        Time        Chart                                Groups
--------------------------------------------------------------------------
ss128       2.51    *********************************    |
ss255       2.51    *********************************    |
ss129       2.58    **********************************   |
--------------------------------------------------------------------------
                        Individual Test Descriptions
--------------------------------------------------------------------------
TYPE color IS ( white, red, yellow, green, blue, brown, black ) ;
hue    : color := yellow ;
--------------------------------------------------------------------------
ss128 IF hue < black THEN hue := color'succ ( hue ) ; END IF ;
      IF hue > white THEN hue := color'pred ( hue ) ; END IF ;
   -- uses 'SUCC and 'PRED on enumerated type, no checking
--------------------------------------------------------------------------
ss129 IF ei < 6 THEN ei := ei + 1 ; END IF ;
      IF ei > 0 THEN ei := ei - 1 ; END IF ;
   -- Same computations as in ss128 on integers .
--------------------------------------------------------------------------
ss255 IF hue < black THEN hue := color'succ ( hue ) ; END IF ;
      IF hue > white THEN hue := color'pred ( hue ) ; END IF ;
   -- uses 'SUCC and 'PRED on enumerated type, enabling range checking
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Bounds Checking (suppression/non-suppression)
--------------------------------------------------------------------------
Test        Execution   Bar                                  Similar
Name        Time        Chart                                Groups
```

```
ss194      3.95    ************************         |
ss175      5.97    **********************************      |
------------------------------------------------------------------
                   Individual Test Descriptions
------------------------------------------------------------------
TYPE a_type IS ARRAY ( 1..10 ) OF int;
a, b, c : a_type := ( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ) ;
minus_one : CONSTANT := -1 ;
d : ARRAY ( minus_one..8 ) OF int := ( -1, 0, 1, 2, 3, 4, 5, 6, 7, 8 ) ;
------------------------------------------------------------
ss175 ii := a ( ei ) + b ( ei ) + c ( ei ) + d ( ei ) ;
   -- Reference to 4 arrays with overlapping static bounds
   -- Can merge bounds checking.
------------------------------------------------------------------
TYPE a_type IS ARRAY ( int' ( 1 ) .. int' ( 10 ) ) OF int ;
a, b, c : a_type := ( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ) ;
minus_one : CONSTANT int := -1 ;
d : ARRAY ( minus_one..int' ( 8 ) ) OF int :=
         ( -1, 0, 1, 2, 3, 4, 5, 6, 7, 8 ) ;
------------------------------------------------------------
ss194 ii := a ( ei ) + b ( ei ) + c ( ei ) + d ( ei ) ;
   -- Reference to 4 arrays.  This version suppresses subscript checking.
------------------------------------------------------------------


------------------------------------------------------------------
Dubious Constant Propagation
------------------------------------------------------------------
      Description                           Optimized?
------------------------------------------------------------------
Time : ss314 (        8.0 ) vs ss315 (        1.5 )    nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss314 => xx := 1.00000001 ; bool := xx - 1.0 > 0.0 ;
         -- test for constant propagation -- precise floating point
         -- literal (9 digits) which can be propagated into its
         -- following statements and folded.
         -- the optimization into ss315 is dubious
      ------------------------------------------------------------
      ss315 => xx := 1.0 ;  bool := true ;
         -- hand optimized (folded) version of ss314
      ------------------------------------------------------------


------------------------------------------------------------------
Dubious Constant Propagation
------------------------------------------------------------------
      Description                           Optimized?
------------------------------------------------------------------
Time : ss316 (        8.8 ) vs ss317 (        1.4 )    nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss316 => xx := 1.00000001 ;  bool := xx - 1.0 > 0.0 ; xx := yy ;
         -- this pair may yield different results than ss314, ss315
         -- since ss316 makes initial assignment to 'xx' dead and
         -- so may facilitate the 'optimization' into ss317
      ------------------------------------------------------------
      ss317 => bool := true ;  xx := yy ;
         -- hand optimized (folded) version of ss316
      ------------------------------------------------------------


------------------------------------------------------------------
Dubious Constant Propagation
------------------------------------------------------------------
      Description                           Optimized?
------------------------------------------------------------------
```

```
Time : ss318 (          1.0 ) vs ss315 (          1.5 )     yes
--------------------------------------------------------------------

        --------------------------------------------------------------
        ss318 => xx := 1.00000001 ;  bool := 1.00000001 - 1.0 > 0.0 ;
           -- use of literal expression could be folded
        --------------------------------------------------------------
        ss315 => xx := 1.0 ;  bool := true ;

-- note, these 5 tests will detect dubious constant
-- propagation of floating point values when there is a single
-- precision float type of 6 or 7 digits (eg. 32 bits) and
-- a more precise type with at least 9 so that the literal
-- expression 1.00000001-1.0 will yield value 1.0e-8 but when
-- stored into single precision variable will have value 0.0.
-- The optimization of propagating the long literal would be
-- dubious, although it would be valid to propagate the value
-- of the literal rounded (or truncated as is systems normal
-- procedure) to single precision.
--------------------------------------------------------------------


--------------------------------------------------------------------
Constant Propagation
--------------------------------------------------------------------
        Descriptior                             Optimized?
--------------------------------------------------------------------
Time : ss366 (          0.8 ) vs ss7 (          0.8 )     nostatistics
--------------------------------------------------------------------

        --------------------------------------------------------------
        ss366 ri := 1 ; -- range check enabled
           -- assign literal to variable with range constraints
           -- Optimization : folding (omit tests at execution time)
        --------------------------------------------------------------
        ss7  kk := 1 ;
           -- Integer assignment, literal "1" to library scope variable.
        --------------------------------------------------------------


--------------------------------------------------------------------
Constant Propagation
--------------------------------------------------------------------
        Description                             Optimized?
--------------------------------------------------------------------
Time : ss540 (          0.8 ) vs ss7 (          0.8 )     nostatistics
--------------------------------------------------------------------

        --------------------------------------------------------------
        ss540 xx := ( 1.0 + 2.0 ** ( -100 ) ) - 1.0 ;
           -- Literal floating point expression in assignment statement.
           -- LRM does not require evaluation with rational package.
           -- Problem also tests precision of evaluation.
        --------------------------------------------------------------
        ss7  kk := 1 ;
           -- Integer assignment, literal "1" to library scope variable.
        --------------------------------------------------------------


--------------------------------------------------------------------
Constant Propagation
--------------------------------------------------------------------
        Description                             Optimized?
--------------------------------------------------------------------
Time : ss556 (          1.6 ) vs ss7 (          0.8 )     nostatistics
--------------------------------------------------------------------

        --------------------------------------------------------------
        ss556 ii := 0 ;
                ii := ii + 1 ;  -- constant propagation?
           -- Integer constant propagation. Assign zero to a variable,
           -- increment variable in next statement.
```

```
-----------------------------------------------------------------------
ss7  kk := 1 ;
   -- Integer assignment, literal "1" to library scope variable.
-----------------------------------------------------------------------


-----------------------------------------------------------------------
Common Subexpression Elimination
-----------------------------------------------------------------------
      Description                                   Optimized?
-----------------------------------------------------------------------
Time : ss210 (      12.0 ) vs ss211 (      16.0 )    yes
-----------------------------------------------------------------------

      -----------------------------------------------------------------
      expression yy*zz is common in ss210 and hand optimized in ss211
      -----------------------------------------------------------------
      ss210 => xx := ( yy * zz - 0.125 ) / ( yy * zz );
      -----------------------------------------------------------------
      ss211 => xx := yy * zz ; xx := ( xx - 0.125 ) / xx ;
   ----------------------------------------------------------------------


-----------------------------------------------------------------------
Common Subexpression Elimination
-----------------------------------------------------------------------
      Description                                   Optimized?
-----------------------------------------------------------------------
Time : ss210 (      12.0 ) vs ss643 (      12.0 )    yes
-----------------------------------------------------------------------

      -----------------------------------------------------------------
      expression yy*zz is common in ss210 and yy*zz and zz*yy are
      common in ss643
      -----------------------------------------------------------------
      ss210 => xx := ( yy * zz - 0.125 ) / ( yy * zz );
      -----------------------------------------------------------------
      ss643 => xx := ( yy * zz - 0.125 ) / ( zz * yy );
   ----------------------------------------------------------------------


-----------------------------------------------------------------------
Common Subexpression Elimination
-----------------------------------------------------------------------
      Description                                   Optimized?
-----------------------------------------------------------------------
Time : ss211 (      16.0 ) vs ss530 (      15.9 )    nostatistics
-----------------------------------------------------------------------

      -----------------------------------------------------------------
      in ss530, x is a local variable not visible in handler
      -----------------------------------------------------------------
      ss211 => xx := yy * zz ; xx := ( xx - 0.125 ) / xx ;
      -----------------------------------------------------------------
      ss530 => x := yy * zz ; xx := ( x - 0.125 ) / x;
   ----------------------------------------------------------------------


-----------------------------------------------------------------------
Common Subexpression Elimination
-----------------------------------------------------------------------
      Description                                   Optimized?
-----------------------------------------------------------------------
Time : ss211 (      16.0 ) vs ss533 (      15.9 )    nostatistics
-----------------------------------------------------------------------

      -----------------------------------------------------------------
      in ss533, y is a local variable not visible in handler
      -----------------------------------------------------------------
      ss211 => xx := yy * zz ; xx := ( xx - 0.125 ) / xx ;
      -----------------------------------------------------------------
      ss533 => y := yy * zz ; xx := ( y - 0.125 ) / y;
   ----------------------------------------------------------------------
```

```
----------------------------------------------------------------
Common Subexpression Elimination
----------------------------------------------------------------
      Description                                 Optimized?
----------------------------------------------------------------
Time : ss644 (          1.1 ) vs ss84 (          0.8 )    nostatistics
----------------------------------------------------------------

      ----------------------------------------------------------
      ss84 IF ll > mm THEN die; END IF; -- False
      ----------------------------------------------------------
      ss644
         IF     ll = mm AND ll = mm AND ll = mm AND ll = mm AND ll = mm
           AND mm = ll AND mm = ll AND mm = ll AND mm = ll AND mm = ll
         THEN
             aie;
         END IF;
----------------------------------------------------------------


----------------------------------------------------------------
Boolean Variable Elimination
----------------------------------------------------------------
      Description                                 Optimized?
----------------------------------------------------------------
Time : ss176 (          4.2 ) vs ss177 (          2.9 )    nostatistics
----------------------------------------------------------------

      ----------------------------------------------------------
      ----------------------------------------------------------
      ss176 bool := ll /= mm ;
            IF bool THEN ii:= 0 ;
                    ELSE ii:= 1 ;
            END IF ;
            bool := True ;
      -- the standardization of ll /= mm and assignment to bool could
      -- be eliminated.  If so, ss176 should have same time as ss177
      ----------------------------------------------------------
      ss177 IF ll /= mm THEN ii := 0 ;
                        ELSE ii := 1 ;
            END IF ;
            bool := True ;
      -- Problem has had boolean variable elimination performed by hand.
----------------------------------------------------------------


----------------------------------------------------------------
Dead Code Elimination
----------------------------------------------------------------
      Description                                 Optimized?
----------------------------------------------------------------
Time : ss56 (          1.6 ) vs ss11 (          0.8 )    nostatistics
----------------------------------------------------------------

      ----------------------------------------------------------
      ss56  ii := ll ;  ii := mm ; -- first assignment is dead
      -- Optimization test for dead assignment elimination on integers.
      ----------------------------------------------------------
      ss11  kk := ll ;
      ----------------------------------------------------------


----------------------------------------------------------------
Dead Code Elimination
----------------------------------------------------------------
      Description                                 Optimized?
----------------------------------------------------------------
Time : ss68 (          1.6 ) vs ss3 (          0.8 )    no
----------------------------------------------------------------
      ----------------------------------------------------------
```

```
        ss68  xx := yy ;  xx := zz ;  -- first assignment is dead
        -- dead assignment elimination; floating point variable
        ------------------------------------------------------------------
        ss3  xx := yy ;
        ------------------------------------------------------------------


------------------------------------------------------------------------
Dead Code Elimination
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss71 (          0.0 ) vs ss0 (          0.0 )     yes
------------------------------------------------------------------------
        ------------------------------------------------------------------
        ss71  xx := xx ;
        -- Assign float variable to itself.
        ------------------------------------------------------------------
        ss0  NULL ;
        ------------------------------------------------------------------


------------------------------------------------------------------------
Dead Code Elimination
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss225 (         4.9 ) vs ss7 (          0.8 )     no
------------------------------------------------------------------------
        ------------------------------------------------------------------
        ss225  FOR i IN int'(1)..int'(5)
               LOOP    ii := i ;
               END LOOP ;
               ii := 0 ;
        -- dead assignments within loop, killed by assignment after exit.
        ------------------------------------------------------------------
        ss7  kk := 1 ;
        ------------------------------------------------------------------


------------------------------------------------------------------------
Dead Code Elimination
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss226 (         0.0 ) vs ss0 (          0.0 )     yes
------------------------------------------------------------------------
        ------------------------------------------------------------------
        ss226 DECLARE
                 xyz : real ;
              BEGIN
                 xyz := yy * zz ;
              END ;
        -- dead assignments within a block.  Variable assigned to
        -- local which is not referenced before block is exited.
        ------------------------------------------------------------------
        ss0  NULL ;
        ------------------------------------------------------------------


------------------------------------------------------------------------
Dead Code Elimination
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss649 (         9.8 ) vs ss1 (          0.8 )     nostatistics
------------------------------------------------------------------------
        ------------------------------------------------------------------
        ss649 IF ll = mm
```

```
              THEN
                 xx := zz * ( one / 2.0 ) ;   -- dead
              ELSE
                 xx := zz * ( one * 0.5 ) ;   -- dead
              END IF ;
              xx := 1.0 ;   -- this kills both assignments in the if
        ------------------------------------------------------------------
        ss1   xx := 1.0 ;
     ------------------------------------------------------------------
```

Dead Code Elimination

```
        Description                             Optimized?
```

```
Time : ss651 (          9.0 ) vs ss11 (          0.8 )      nostatistics
```

```
        ss651 FOR i IN 1..10
              LOOP
                 kk := i ;
              END LOOP ;
              ii := kk ;
        -- Assign to variable within a loop, after loop exit,
        -- making all assignments within the loop dead.
        ------------------------------------------------------------------
        ss11  kk := 11 ;
     ------------------------------------------------------------------
```

Dead Code Elimination

```
        Description                             Optimized?
```

```
Time : ss93 (          0.0 ) vs ss0 (          0.0 )        yes
```

```
        ss93 IF False THEN die ;  END IF ;
        -- redundant code elimination - could be noop
        ------------------------------------------------------------------
        ss0    NULL  ;
     ------------------------------------------------------------------
```

Dead Code Elimination

```
        Description                             Optimized?
```

```
Time : ss195 (          0.0 ) vs ss0 (          0.0 )        yes
```

```
        ss195 ii := ii ;
        -- superfluous integer assignment
        ------------------------------------------------------------------
        ss0    NULL  ;
     ------------------------------------------------------------------
```

Dead Code Elimination

```
        Description                             Optimized?
```

```
Time : ss261 (          0.0 ) vs ss0 (          0.0 )        yes
```

```
      ss261 GOTO Label ;  << Label>> NULL ;
      -- Peephole optimizer should translate into a NULL.
      ---------------------------------------------------------------------
      ss0   NULL  ;
   ---------------------------------------------------------------------------
```

Dead Code Elimination

Description	Optimized?
Time : ss26 (0.3) vs ss0 (0.0)	nostatistics

```
      ---------------------------------------------------------------------
      ss26  GOTO 12 ;
            <<l1>> die ;
            GOTO l1 ;
            <<l2>> NULL ;
      -- Language feature test, GOTO.
      ---------------------------------------------------------------------
      ss0   NULL  ;
   ---------------------------------------------------------------------------
```

Dead Code Elimination

Description	Optimized?
Time : ss376 (2.2) vs ss36 (2.3)	yes

```
      ---------------------------------------------------------------------
      ss376 LOOP   proc0 ;   EXIT ;   END LOOP ;
      -- example with simplifiable control flow
      -- redundant code elimination - (omit LOOP)
      ---------------------------------------------------------------------
      ss36 proc0 ;
      -- call to library scope procedure -- body is null.
      ---------------------------------------------------------------------
```

Dead Code Elimination

Description	Optimized?
Time : ss377 (2.4) vs ss36 (2.3)	no

```
      ---------------------------------------------------------------------
      ss377 LOOP
            proc0 ;
            EXIT ;
            xx := e3 ( ei, ej, ek ) ;
         END LOOP ;
      -- assignment is unreachable, LOOP EXIT code is superfluous
      ---------------------------------------------------------------------
      ss36 proc0 ;
      -- call to library scope procedure -- body is null.
      ---------------------------------------------------------------------
```

Dead Code Elimination

Description	Optimized?
Time : ss543 (0.3) vs ss0 (0.0)	nostatistics

```
        ss543 DECLARE -- should be noop
            BEGIN
                NULL ; -- null body, no way to enter the handler
            EXCEPTION
                WHEN OTHERS =>
                    die ; -- no re-raise in handler
            END ;
        -- Declare block with null body and exception handler,
        -- which is unreachable and superfluous.
        -- Optimization    : unreachable code elimination
```

```
        ss0  null;
        -- Language feature test, null statement
```

Dead Code Elimination

Description	Optimized?
Time : ss544 (0.0) vs ss0 (0.0) yes	

```
        ss544 DECLARE -- should be noop
            BEGIN
                NULL ;
            END ;
        -- null body check for block overhead
```

```
        ss0  null;
        -- Language feature test, null statement
```

Dead Code Elimination

Description	Optimized?
Time : ss751 (0.0) vs ss0 (0.0) yes	

```
        ss751 IF False
            THEN
                ii := jj ;
            END IF ; -- unreachable assignment
        -- Optimization test:  omission of an unreachable assignment.
```

```
        ss0  null;
        -- Language feature test, null statement
```

Dead Variable Elimination

Test Name	Execution Time	Bar Chart	Similar Groups
ss639	8.76	***********************	|
ss640	11.90	*********************************	|
ss638	12.80	***********************************	|

Individual Test Descriptions

```
ss638 DECLARE
        state : int ;   -- all are live due to handler
      BEGIN
        state := 1;         proc0;
        state := 2;         proc0;
        state := 3;         proc0;
        state := 4;         proc0;
      EXCEPTION
        WHEN OTHERS =>
           procil ( state ) ;
           die;
      END;
-- comparison to ss639 to check for dead variable elimination.
-----------------------------------------------------------------
ss639 DECLARE
        state : int; -- assignments are dead due to no handler
      BEGIN
        state := 1;         proc0;
        state := 2;         proc0;
        state := 3;         proc0;
        state := 4;         proc0;
      END;
-- dead variable elimination. State never referenced.
-----------------------------------------------------------------
ss640 DECLARE -- assignments to global, force live
      BEGIN
        ii := 1;           proc0;
        ii := 2;           proc0;
        ii := 3;           proc0;
        ii := 4;           proc0;
      END ;
-- comparison for dead variable elimination. ii is global
-----------------------------------------------------------------


-----------------------------------------------------------------
Order Of Evaluation Tests
-----------------------------------------------------------------
      Description                                  Optimized?
-----------------------------------------------------------------
Time : ss413 (        12.7 ) vs ss414 (        12.9 )    yes
-----------------------------------------------------------------

      -----------------------------------------------------------
      ss413 xx := sgn ( yy ) + 1.7 ;
      -- order of evaluation test
      -----------------------------------------------------------
      ss414 xx := 1.7 + sgn ( yy ) ;
      -- order of evaluation test.  A simple left-to-right order of
      -- evaluation would load the literal, save value when it calls
      -- on the function, and restore it after the function call.
      -----------------------------------------------------------


-----------------------------------------------------------------
Order Of Evaluation Tests
-----------------------------------------------------------------
      Description                                  Optimized?
-----------------------------------------------------------------
Time : ss415 (        11.1 ) vs ss416 (        11.5 )    yes
-----------------------------------------------------------------

      -----------------------------------------------------------
                        -- computes the square root of 2.0
                        -- statement is a Newton iteration
      xx := 1.414159 ;  -- this makes each iteration essentially
                        -- a self transformation
      -----------------------------------------------------------
      ss415 xx := 0.5 * ( xx + 2.0 / xx ) ;
```

```
            -- order of evaluation test, simple left-to-right order of
            -- evaluation will load variable and then have to do a
            -- register to register operation to add right hand
            -- subexpression.
    ------------------------------------------------------------------
            ss416 xx := 0.5 * ( 2.0 / xx + xx ) ;
            -- order of evaluation test, simple left-to-right order of
            -- evaluation of subexpression is best here (perform
            -- the divide and then add from memory - no need to save
            -- and restore the quotient - however, the multiply by
            -- 0.5 should be deferred).
    ------------------------------------------------------------------


    ------------------------------------------------------------------
    Data Flow
    ------------------------------------------------------------------
            Description                              Optimized?
    ------------------------------------------------------------------
    Time : ss427 (         1.8 ) vs ss11 (       0.8 )     nostatistics
    ------------------------------------------------------------------

            ss427  => LOOP ii := 11 ; EXIT WHEN ii = 11 ; die ;  END LOOP ;
               -- Assign integer to another integer and test if the two are equal.
    ------------------------------------------------------------------
            ss11  => kk := 11 ;
               -- Library scope integer assignment.
    ------------------------------------------------------------------


    ------------------------------------------------------------------
    Data Flow
    ------------------------------------------------------------------
            Description                              Optimized?
    ------------------------------------------------------------------
    Time : ss504 (         0.0 ) vs ss0 (         0.0 )     yes
    ------------------------------------------------------------------

            ss504  => IF kk /= kk THEN die ; END IF ;
               -- kk /= kk is foldable to false.
    ------------------------------------------------------------------
            ss0   => NULL ;
    ------------------------------------------------------------------


    ------------------------------------------------------------------
    Data Flow
    ------------------------------------------------------------------
            Description                              Optimized?
    ------------------------------------------------------------------
    Time : ss505 (         1.6 ) vs ss0 (         0.0 )     nostatistics
    ------------------------------------------------------------------

            ss505  => IF ii <= 2 AND ii > 2 THEN die ; END IF ;
               -- foldable into false. No value could satisfy both subexpressions.
    ------------------------------------------------------------------
            ss0   => NULL ;
    ------------------------------------------------------------------


    ------------------------------------------------------------------
    Data Flow
    ------------------------------------------------------------------
            Description                              Optimized?
    ------------------------------------------------------------------
    Time : ss558 (         2.6 ) vs ss559 (       2.5 )     nostatistics
    ------------------------------------------------------------------

            ss558 => IF mm = 3 AND 11 = 2    -- true
```

```
                 THEN   ii := mm - ll ;    -- could fold into 'ii := 1;'
                 END IF ;
         -- if variables did not have values of 3 and 2, respectively
         -- then the alternative would not execute, therefore optimizer
         -- can simplify expressions by using bounds determined by relations.
         ------------------------------------------------------------------
         ss559 => IF mm = 3 AND ll = 2   -- true
                 THEN   ii := 1;
                 END IF ;
      ------------------------------------------------------------------
```

Data Flow

```
-------------------------------------------------------------------------
       Description                                    Optimized?
-------------------------------------------------------------------------
Time : ss756 (         0.8 ) vs ss7 (         0.8 )         no
-------------------------------------------------------------------------
```

```
      ------------------------------------------------------------------
      ss756 => fold testing of range constraint test which is in range

            BEGIN
                ri := 0;         -- range on ri is -2..2,
            EXCEPTION            -- range verification at compile time?
                WHEN Constraint_error => die;    -- never reached
            END;
         -- Could be translated as simple assignment
         -- Range checking would be verified at compile time
         ------------------------------------------------------------------
      ss7  => kk := 1 ;
         -- Integer literal assignment to library scope variable.
      ------------------------------------------------------------------
```

Data Flow

Test Name	Execution Time	Bar Chart	Similar Groups
ss753	147.00	**********************************	|
ss754	147.00	**********************************	|
ss755	147.50	**********************************	|

Individual Test Descriptions

```
 ------------------------------------------------------------------
 ss753   BEGIN
             ri := real'mantissa; -- range on ri is -2..2,
                                  -- real'mantissa needs 6 digits
             die;                 -- will never be reached
         EXCEPTION
             WHEN Constraint_error => proc0;
         END ;
      -- Could translate into simple call on ""proc0"" since the control
      -- path is determinable at compile time.  Assign out of range static
      -- expression to an integer with range constraints.  See if it
      -- optimizes into a simple raise of CONSTRAINT_ERROR.
 ------------------------------------------------------------------
 ss754   BEGIN
             IF real'mantissa NOT IN -2..2
             THEN
                 RAISE Constraint_error;
             END IF;
             ri := real'mantissa; -- range on ri is -2..2
             die;                 -- will never be reached
```

```
        EXCEPTION
            WHEN Constraint_error => proc0;
        END ;
        -- Could translate into simple call on ""proc0"" since the control
        -- path is determinable at compile time.  Explicit IF statement tests
        -- static expression out of range and raises CONSTRAINT_ERROR.
   ----------------------------------------------------------------------
 ss755  BEGIN
            ri := real'mantissa; -- range on ri is -2..2,
                                 -- contraint_error will be raised here
            die;                 -- never reached
        EXCEPTION
            WHEN Constraint_error => NULL; -- this path taken
        END ;
        -- Could be translated as null.  Assign out-of-range static
        -- expression to a variable with range constraints.  Null handler.
   ----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Folding
----------------------------------------------------------------------
      Description                                  Optimized?
----------------------------------------------------------------------
Time : ss40 (         1.6 ) vs ss11 (        0.8 )     nostatistics
----------------------------------------------------------------------
        ----------------------------------------------------------------------
        ss40 ii := -11 ;
        -- integer unary minus.
        ----------------------------------------------------------------------
        ss11 kk := 11 ;
        -- Library scope integer assignment.
        ----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Folding
----------------------------------------------------------------------
      Description                                  Optimized?
----------------------------------------------------------------------
Time : ss41 (         0.8 ) vs ss7 (        0.8 )      no
----------------------------------------------------------------------
        ----------------------------------------------------------------------
        ss41 ii := 1 + 1 ;
        -- test for folding of static integer expression, "1+1".
        ----------------------------------------------------------------------
        ss7  kk := 1 ;
        -- Integer assignment, literal "1" to library scope variable.
        ----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Folding
----------------------------------------------------------------------
      Description                                  Optimized?
----------------------------------------------------------------------
Time : ss42 (         0.8 ) vs ss7 (        0.8 )     nostatistics
----------------------------------------------------------------------
        ----------------------------------------------------------------------
        ss42 ii := -1 ;
        -- test for folding of static integer expression, "-1".
        ----------------------------------------------------------------------
        ss7  kk := 1 ;
        -- Integer assignment, literal "1" to library scope variable.
        ----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Folding
```

```
------------------------------------------------------------------
      Description                                   Optimized?
------------------------------------------------------------------
Time : ss216 (          7.5 ) vs ss1 (         0.8 )    nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss216 xx := 2.0 ;   xx := xx ** 2 ;
      -- example floating point, constant folding, constant propagating
      ------------------------------------------------------------
      ss1  xx := 1.0 ;
      -- Assign floating point variab'e from literal vaiue.
      ---------------------------------------------------------


------------------------------------------------------------------
Folding
------------------------------------------------------------------
      Description                                   Optimized?
------------------------------------------------------------------
Time : ss217 (          3.5 ) vs ss7 (         0.8 )    nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss217 ii := 2 ;   ii := ii ** 2 ; -- could be folded into ii := 4 ;
      -- example integer point constant folding, constant propagating
      ------------------------------------------------------------
      ss7  kk := 1 ;
      -- Integer literal assignment to library scope variable.
      ---------------------------------------------------------


------------------------------------------------------------------
Folding
------------------------------------------------------------------
      Description                                   Optimized?
------------------------------------------------------------------
Time : ss219 (          0.8 ) vs ss1 (         0.8 )    nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss219 xx := 2.0 ** 2 ;
      -- foldable floating point expression. Equivalent to ss216.
      ------------------------------------------------------------
      ss1  xx := 1.0 ;   5.4
      -- Assign floating point variable from literal value.
      ---------------------------------------------------------


------------------------------------------------------------------
Folding
------------------------------------------------------------------
      Description                                   Optimized?
------------------------------------------------------------------
Time : ss303 (          1.6 ) vs ss302 (        1.6 )    yes
------------------------------------------------------------------

      ------------------------------------------------------------
      ss303 dx := double ( 1 ) ;
        -- convert integer literal to double
      ------------------------------------------------------------
      ss302 dx := 1.0 ;
        -- extended precision floating point literal assignment
      ---------------------------------------------------------


------------------------------------------------------------------
Folding
------------------------------------------------------------------
      Description                                   Optimized?
------------------------------------------------------------------
Time : ss304 (         33.2 ) vs ss307 (       27.1 )    nostatistics
------------------------------------------------------------------
```

```
   ----------------------------------------------------------------
   ss304 xx := yy ** 16 ;
      -- floating point exponentiation,  ** 16
   ----------------------------------------------------------------
   ss307 xx := yy * yy ;       xx := xx * xx ;
         xx := xx * xx ;       xx := xx * xx ;
      -- floating point exponentiation comparison
   ----------------------------------------------------------------
```

```
-----------------------------------------------------------------------
Folding
-----------------------------------------------------------------------
      Description                                Optimized?
-----------------------------------------------------------------------
Time : ss305 (        41.1 ) vs ss307 (        27.1 )    nostatistics
-----------------------------------------------------------------------
```

```
   ----------------------------------------------------------------
   ss305 xx := (yy ** 4) ** 4 ;
      -- floating point exponentiation,  ** 4) ** 4
   ----------------------------------------------------------------
   ss307 xx := yy * yy ;       xx := xx * xx ;
         xx := xx * xx ;       xx := xx * xx ;
      -- floating point exponentiation comparison
   ----------------------------------------------------------------
```

```
-----------------------------------------------------------------------
Folding
-----------------------------------------------------------------------
      Description                                Optimized?
-----------------------------------------------------------------------
Time : ss532 (         7.5 ) vs ss529 (         7.5 )    yes
-----------------------------------------------------------------------
```

```
   ----------------------------------------------------------------
   ss532  y := 2.0 ;  xx := y ** 2 ;
      -- constant propagating : local variable visible in handler
   ----------------------------------------------------------------
   ss529 x := 2.0 ;xx := x ** 2 ;
      -- cf ss216
      -- constant propagating with local variable not visible in handler
   ----------------------------------------------------------------
```

```
-----------------------------------------------------------------------
Folding
-----------------------------------------------------------------------
      Description                                Optimized?
-----------------------------------------------------------------------
Time : ss532 (         7.5 ) vs ss1 (         0.8 )    nostatistics
-----------------------------------------------------------------------
```

```
   ----------------------------------------------------------------
   ss532  y := 2.0 ;  xx := y ** 2 ;
      -- constant propagating : local variable visible in handler
   ----------------------------------------------------------------
   ss1  xx := 1.0 ;
      -- Assign floating point variable from literal value.
   ----------------------------------------------------------------
```

```
-----------------------------------------------------------------------
Folding
-----------------------------------------------------------------------
      Description                                Optimized?
-----------------------------------------------------------------------
Time : ss2 (         0.8 ) vs ss1 (         0.8 )    nostatistics
-----------------------------------------------------------------------
   ----------------------------------------------------------------
   ss2  xx := real ( 1 ) ;
```

```
      -- Type conversion in static expression -- real ( 1 ) .
---------------------------------------------------------------------
ss1   xx := 1.0 ;
-- Assign floating point variable from literal value.
---------------------------------------------------------------------


---------------------------------------------------------------------
Folding
---------------------------------------------------------------------
       Description                                  Optimized?
---------------------------------------------------------------------
Time : ss8 (         0.8 ) vs ss7 (         0.8 )     yes
---------------------------------------------------------------------

      ---------------------------------------------------------------
ss8   kk := int ( 1.0 ) ;
-- Type conversion from floating point literal to integer.
---------------------------------------------------------------------
ss7   kk := 1;
-- Integer assignment, literal "1" to library scope variable.
---------------------------------------------------------------------


---------------------------------------------------------------------
Folding
---------------------------------------------------------------------
       Description                                  Optimized?
---------------------------------------------------------------------
Time : ss54 (       1.4 ) vs ss53 (       1.3 )     nostatistics
---------------------------------------------------------------------

      ---------------------------------------------------------------
ss54 ii := i1 ( ei + 1 ) ;
-- Reference to subscripted array of int, without checking.
-- Optimization :  fold constant term into addressing expression
---------------------------------------------------------------------
ss53 ii := i1 ( ei ) ;
-- Reference to subscripted array of int, without checking.
---------------------------------------------------------------------


---------------------------------------------------------------------
Folding
---------------------------------------------------------------------
       Description                                  Optimized?
---------------------------------------------------------------------
Time : ss55 (       0.8 ) vs ss11 (       0.8 )     yes
---------------------------------------------------------------------

      ---------------------------------------------------------------
ss55 ii := i1 ( 1 ) ;
-- Reference array with a constant subscript.
---------------------------------------------------------------------
ss11 kk := 11 ;
-- Library scope integer assignment.
---------------------------------------------------------------------


---------------------------------------------------------------------
Folding
---------------------------------------------------------------------
       Description                                  Optimized?
---------------------------------------------------------------------
Time : ss60 (       0.8 ) vs ss1 (       0.8 )     nostatistics
---------------------------------------------------------------------

      ---------------------------------------------------------------
ss60 xx := -1.0 ;     -- fold minus into literal value
-- Assign negative floating literal to scalar.
---------------------------------------------------------------------
ss1   xx :=  1.0;
-- Assign floating point variable from literal value.
```

```
    ------------------------------------------------------------------

    ------------------------------------------------------------------
Folding
    ------------------------------------------------------------------
      Description                                      Optimized?
    ------------------------------------------------------------------
Time : ss189 (          5.6 ) vs ss190 (          4.8 )    nostatistics
    ------------------------------------------------------------------

       ------------------------------------------------------------
       ss189 ii := -mm / 3 ; -- can rewrite as ii := mm / -3 ;
       -- Could fold leading unary minus into a literal
       ------------------------------------------------------------
       ss190 ii := mm / (-3) ;
       -- Hand folded version of ss189.
    ------------------------------------------------------------------


    ------------------------------------------------------------------
Foldable Expressions
    ------------------------------------------------------------------
      Description                                      Optimized?
    ------------------------------------------------------------------
Time : ss587 (         21.3 ) vs ss594 (          6.0 )    no
    ------------------------------------------------------------------
       x1 : CONSTANT := 2.0 ** (-70) ;
       x2 : CONSTANT  real := 2.0 ** (-70) ;
       x3 : real := 2.0 ** (-70) ;
       x4 : real := sgn ( one ) * 2.0 ** (-70) ;
       x5 : real := sgn ( one ) * 2.0 ** (-70) ;
       x0 : CONSTANT := 10.0 * x1 ;
       y  : real := 2.0 ** (+70) ;
     ---------------------------------------------------------------

     ---------------------------------------------------------------
       ss587 y := y + x1 + x1 + x1 + x1 + x1 + x1 + x1 + x1 + x1 + x1 ;
       -- for precisions less than 140 bits,
       -- ""y"" is unchanged by addition, term too small
       -- expression with foldable subexpression using named number
       ---------------------------------------------------------------
       ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
       -- comparison with ss587 - 593, hand folded version
       ---------------------------------------------------------------


    ------------------------------------------------------------------
Foldable Expressions
    ------------------------------------------------------------------
      Description                                      Optimized?
    ------------------------------------------------------------------
Time : ss588 (         21.3 ) vs ss594 (          6.0 )    no
    ------------------------------------------------------------------

       ------------------------------------------------------------
       ss588 y := y + x2 + x2 + x2 + x2 + x2 + x2 + x2 + x2 + x2 + x2 ;
       -- foldable subexpression using constant real
       ------------------------------------------------------------
       ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
       -- comparison with ss587 - 593, hand folded version
       ------------------------------------------------------------


    ------------------------------------------------------------------
Foldable Expressions
    ------------------------------------------------------------------
      Description                                      Optimized?
    ------------------------------------------------------------------
Time : ss589 (         24.7 ) vs ss594 (          6.0 )    nostatistics
    ------------------------------------------------------------------

       ------------------------------------------------------------
```

```
ss589 y := y + x3 + x3 + x3 + x3 + x3 + x3 + x3 + x3 + x3 + x3 ;
-- using variable initialized with literal and not modified
-----------------------------------------------------------------
ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
-- comparison with ss587 - 593, hand folded version
-----------------------------------------------------------------
```

Foldable Expressions

Description	Optimized?

Time : ss590 (24.8) vs ss594 (6.0)   no

```
ss590 y := y + x4 + x4 + x4 + x4 + x4 + x4 + x4 + x4 + x4 + x4 ;
-- expression with foldable subexpression using variable
-- initialized with expression and not modified
-----------------------------------------------------------------
ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
-- comparison with ss587 - 593, hand folded version
-----------------------------------------------------------------
```

Foldable Expressions

Description	Optimized?

Time : ss591 (24.7) vs ss594 (6.0)   no

```
ss591 y := y + x5 + x5 + x5 + x5 + x5 + x5 + x5 + x5 + x5 + x5 ;
-- comparison with ss587-590,  using variable which is
-- modified, but is invariant within the timing loop
-----------------------------------------------------------------
ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
-- comparison with ss587 - 593, hand folded version
-----------------------------------------------------------------
```

Foldable Expressions

Description	Optimized?

Time : ss592 (25.3) vs ss594 (6.0)   nostatistics

```
ss592 y := y + x5 + x5 + x5 + x5 + x5 + x5 + x5 + x5 + x5 + x5 ;
          IF bool THEN die ;  proc1 (x5 ) ;  END IF ;
--  here expression with x5 is not loop invariant
-- comparison with ss587-591, expression not timing loop invariant
-----------------------------------------------------------------
ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
-- comparison with ss587 - 593, hand folded version
-----------------------------------------------------------------
```

Foldable Expressions

Description	Optimized?

Time : ss593 (24.8) vs ss594 (6.0)   no

```
ss593 y := y + xx + xx + xx + xx + xx + xx + xx + xx + xx + xx ;
```

```
        -- xx is global, expression is not loop invariant
        -- comparison with ss587-592, variable is global, not invariant
        -----------------------------------------------------------------
        ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
        -- comparison with ss587 - 593, hand folded version
        -----------------------------------------------------------------
```

```
-------------------------------------------------------------------------
Foldable Expressions
-------------------------------------------------------------------------
      Description                              Optimized?
-------------------------------------------------------------------------
Time : ss595 (        23.0 ) vs ss594 (        6.0 )     no
-------------------------------------------------------------------------
```

```
        -----------------------------------------------------------------
        ss595 y := y + 2.0 ** (-70) + 2.0 ** (-70) + 2.0 ** (-70) +
                   2.0 ** (-70) + 2.0 ** (-70) + 2.0 ** (-70) +
                   2.0 ** (-70) + 2.0 ** (-70) + 2.0 ** (-70) +
                   2.0 ** (-70) + 2.0 ** (-70) ;
        -- foldable subexpression using literals -- cf ss587
        -----------------------------------------------------------------
        ss594 y := y + x0 ;   -- x0 : constant := 10.0 * x1 ;
        -- comparison with ss587 - 593, hand folded version
        -----------------------------------------------------------------
```

```
-------------------------------------------------------------------------
Foldable Boolean Expressions
-------------------------------------------------------------------------
      Description                              Optimized?
-------------------------------------------------------------------------
Time : ss227 (        1.1 ) vs ss84 (        0.8 )     nostatistics
-------------------------------------------------------------------------
```

```
        -----------------------------------------------------------------
        ss227 IF ll > mm OR False THEN die ;  END IF ;
        -- foldable boolean expression, OR False
        -----------------------------------------------------------------
        ss84  IF ll > mm THEN die ; END IF ;   -- False
        -- IF statement, integer relation (False), no ELSE clause
        -----------------------------------------------------------------
```

```
-------------------------------------------------------------------------
Foldable Boolean Expressions
-------------------------------------------------------------------------
      Description                              Optimized?
-------------------------------------------------------------------------
Time : ss230 (        1.1 ) vs ss84 (        0.8 )     nostatistics
-------------------------------------------------------------------------
```

```
        -----------------------------------------------------------------
        ss230 IF ll > mm OR False OR False THEN die ;  END IF ;
        -- foldable boolean expression, OR False OR False
        -----------------------------------------------------------------
        ss84  IF ll > mm THEN die ; END IF ;   -- False
        -- IF statement, integer relation (False), no ELSE clause
        -----------------------------------------------------------------
```

```
-------------------------------------------------------------------------
Foldable Boolean Expressions
-------------------------------------------------------------------------
      Description                              Optimized?
-------------------------------------------------------------------------
Time : ss231 (        1.5 ) vs ss84 (        0.8 )     nostatistics
-------------------------------------------------------------------------
```

```
        -----------------------------------------------------------------
        ss231 IF ll > mm OR ELSE False OR ELSE False THEN die ; END IF ;
        -- foldable boolean expression OR ELSE False OR ELSE False
```

```
-------------------------------------------------------------------
   ss84  IF ll > mm THEN die ; END IF ;  -- False
   -- IF statement, integer relation (False), no ELSE clause
   -------------------------------------------------------------
```

Foldable Boolean Expressions
```
-------------------------------------------------------------------
     Description                              Optimized?
-------------------------------------------------------------------
Time : ss232 (        1.2 ) vs ss84 (       0.8 )    nostatistics
-------------------------------------------------------------------
   -------------------------------------------------------------
   ss232 IF ll > mm OR ELSE False THEN die ; END IF ;
   -- example of foldable boolean expression OR ELSE False
   -------------------------------------------------------------
   ss84  IF ll > mm THEN die ; END IF ;  -- False
   -- IF statement, integer relation (False), no ELSE clause
   -------------------------------------------------------------
```

Foldable Boolean Expressions
```
-------------------------------------------------------------------
     Description                              Optimized?
-------------------------------------------------------------------
Time : ss239 (        0.0 ) vs ss0 (        0.0 )    yes
-------------------------------------------------------------------
   -------------------------------------------------------------
   ss239 FOR i IN int'(2)..int'(1) LOOP die ; END LOOP ;
   -- example of FOR loop with null range, compile time determinable
   -------------------------------------------------------------
   ss0   NULL ;
   -------------------------------------------------------------
```

Folding in Inline Function
```
-------------------------------------------------------------------
Test      Execution   Bar                               Similar
Name      Time        Chart                             Groups
-------------------------------------------------------------------
ss565        1.42     ******                            |
ss563        1.43     ******                             |
ss564        1.91     ********                             |
ss142        8.19     **********************************    |
-------------------------------------------------------------------
                  Individual Test Descriptions
-------------------------------------------------------------------
  FUNCTION max2 ( i , j : int ) RETURN int IS
  BEGIN
     IF i > j THEN RETURN i ;
             ELSE RETURN j ;
     END IF ;
  END max2 ;
  PRAGMA inline ( max2 ) ;
  ---------------------------------------------------------------
   ss142 xx := max2 ( yy , zz ) ;
     -- call on local inline function
  ---------------------------------------------------------------
   ss563 ii := max2 ( 1 , 100 ) ;
     -- inline function with literals, can be folded into ii := 100
  ---------------------------------------------------------------
   ss564 ii := max2 ( 0 , ei ) ;
     -- inline function test, first actual parameter to max
     -- function is 0, permitting simplification.
  ---------------------------------------------------------------
```

```
       ss565 ii := max2 ( ei , 0 ) ;
          -- inline function test, second actual parameter to max
          -- function is 0, permitting simplification.
-------------------------------------------------------------------------


-------------------------------------------------------------------------
Folding in Inline Function
-------------------------------------------------------------------------
       Description                            Optimized?
-------------------------------------------------------------------------
Time : ss563 (         1.4 ) vs ss7 (       0.8 )      nostatistics
-------------------------------------------------------------------------

       ss563 ii := max2 ( 1 , 100 ) ;
          -- inline function with literals, can be folded into ii := 100
       ss7  kk := 1 ;
          -- Integer literal assignment to library scope variable.
-------------------------------------------------------------------------


-------------------------------------------------------------------------
Machine Idioms
-------------------------------------------------------------------------
       Description                            Optimized?
-------------------------------------------------------------------------
Time : ss29 (         4.0 ) vs ss3 (        0.8 )      no
-------------------------------------------------------------------------

       ss29  xx := abs ( yy ) ;
       -- Language feature test, floating point "abs".
       -- Optimization : machine idiom "load absolute value" instruction
-------------------------------------------------------------------------
       ss3   xx := yy ;
       -- Assignment of two floating point variables, library scope.
-------------------------------------------------------------------------


-------------------------------------------------------------------------
Machine Idioms
-------------------------------------------------------------------------
       Description                            Optimized?
-------------------------------------------------------------------------
Time : ss30 (         0.8 ) vs ss11 (       0.8 )      nostatistics
-------------------------------------------------------------------------

       ss30  ii := abs ( 11 ) ;
       -- Language feature test, integer "abs" .
       -- Optimization : machine idiom, "load absolute value"
-------------------------------------------------------------------------
       ss11  kk := 11 ;
       -- Library scope integer assignment.
-------------------------------------------------------------------------


-------------------------------------------------------------------------
Machine Idioms
-------------------------------------------------------------------------
       Description                            Optimized?
-------------------------------------------------------------------------
Time : ss40 (         1.6 ) vs ss11 (       0.8 )      nostatistics
-------------------------------------------------------------------------

       ss40  ii := -11 ;
       -- Language feature test, integer unary minus.
       -- Optimization     :   machine idiom - "load complement"
-------------------------------------------------------------------------
       ss11  kk := 11 ;
       -- Library scope integer assignment.
```

```
------------------------------------------------------------------

------------------------------------------------------------------
Machine Idioms
------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------
Time : ss45 (         0.8 ) vs ss7 (         0.8 )      nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss45  ii := 0 ;
      -- Assign external integer to zero: machine idiom (clear memory)
      ------------------------------------------------------------
      ss7   kk := 1;
      -- assignment of literal "1" to library scope variable.
      ------------------------------------------------------------


------------------------------------------------------------------
Machine Idioms
------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------
Time : ss52 (         0.9 ) vs ss9 (         1.1 )      yes
------------------------------------------------------------------

      ------------------------------------------------------------
      ss52  ii := 11 + 1 ;
      -- Test use of "INC" instruction for "+1".
      ------------------------------------------------------------
      ss9   kk := 11 + mm ;            -- Integer addition.
      ------------------------------------------------------------


------------------------------------------------------------------
Machine Idioms
------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------
Time : ss59 (         4.0 ) vs ss3 (         0.8 )      nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss59 xx := -yy ;
      -- Unary minus, floating point: machine idiom - load negative
      ------------------------------------------------------------
      ss3   xx := yy ;
      -- Assignment of two floating point variables, library scope.
      ------------------------------------------------------------


------------------------------------------------------------------
Machine Idioms
------------------------------------------------------------------
      Description                                    Optimized?
------------------------------------------------------------------
Time : ss115 (        4.0 ) vs ss114 (        3.1 )     nostatistics
------------------------------------------------------------------

      ------------------------------------------------------------
      ss115 a.field_1 := b.field_1 ;  a.field_3 := b.field_3 ;
            a.field_4 := b.field_4 ;  a.field_2 := b.field_2 ;
      -- Record component by component assignment (all fields).
      -- Optimization: machine idiom (block move instruction)
      ------------------------------------------------------------
      ss114 a := b ;
      -- Record assignment.
      ------------------------------------------------------------


------------------------------------------------------------------
Machine Idioms
```

```
    ------------------------------------------------------------------------
         Description                                    Optimized?
    ------------------------------------------------------------------------
    Time : ss128 (         2.5 ) vs ss129 (        2.6 )    yes
    ------------------------------------------------------------------------

         ------------------------------------------------------------------
         ss128 IF hue < black THEN hue := color'succ(hue) ;   END IF ;
              IF hue > white THEN hue := color'pred(hue) ;   END IF ;
         -- PRED and SUCC functions on enumeration types.
         ------------------------------------------------------------------
         ss129 IF ei < 6 THEN ei := ei + 1 ;   END IF ;
              IF ei > 0 THEN ei := ei - 1 ;   END IF ;
         -- Same computations as in ss128 on integers .
         -- Optimization      :   machine idioms - (inc and dec)
         ------------------------------------------------------------------


    ------------------------------------------------------------------------
    Machine Idioms
    ------------------------------------------------------------------------
         Description                                    Optimized?
    ------------------------------------------------------------------------
    Time : ss196 (         0.9 ) vs ss201 (        2.9 )    yes
    ------------------------------------------------------------------------

         ------------------------------------------------------------------
         ss196 ii := pp * 2 ;
         -- natural integer multiplication: machine idiom - shift?
         ------------------------------------------------------------------
         ss201 ii := pp * 1009 ;
         -- natural integer multiplication - not power of 2
         ------------------------------------------------------------------


    ------------------------------------------------------------------------
    Machine Idioms
    ------------------------------------------------------------------------
         Description                                    Optimized?
    ------------------------------------------------------------------------
    Time : ss198 (         1.1 ) vs ss201 (        2.9 )    yes
    ------------------------------------------------------------------------

         ------------------------------------------------------------------
         ss198 ii := pp * 4 ;
         -- natural integer multiplication: machine idiom - shift?
         ------------------------------------------------------------------
         ss201 ii := pp * 1009 ;
         -- natural integer multiplication - not power of 2
         ------------------------------------------------------------------


    ------------------------------------------------------------------------
    Machine Idioms
    ------------------------------------------------------------------------
         Description                                    Optimized?
    ------------------------------------------------------------------------
    Time : ss201 (         2.9 ) vs ss202 (        3.0 )    yes
    ------------------------------------------------------------------------

         ------------------------------------------------------------------
         ss201 ii := pp * 1009 ;
         -- natural integer multiplication - not power of 2
         ------------------------------------------------------------------
         ss202 ii := 11 * mm ;
         -- integer multiplication
         ------------------------------------------------------------------


    ------------------------------------------------------------------------
    Machine Idioms
    ------------------------------------------------------------------------
         Description                                    Optimized?
```

```
--------------------------------------------------------------------
Time : ss197 (          1.3 ) vs ss203 (          4.8 )     yes
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss197 ii := pp / 2 ;    -- could shift
      -- natural divide multiplication, /2
      -- Optimization      :  machine idiom - shift?
      --------------------------------------------------------------
      ss203 ii := pp / 1009 ;
      -- natural division, 1009
   -----------------------------------------------------------------


--------------------------------------------------------------------
Machine Idioms
--------------------------------------------------------------------
      Description                               Optimized?
--------------------------------------------------------------------
Time : ss199 (          1.1 ) vs ss200 (          1.7 )     yes
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss199 ii := pp mod 4 ;                   -- could mod by masking
      -- natural integer mod, MOD 4
      -- Optimization      :  machine idiom - AND masking operation
      --------------------------------------------------------------
      ss200 ii := pp - ( ( pp / 4 ) * 4 ) ;
      -- expression comparable to MOD 4
      -- Optimization      :  machine idiom - AND masking operation
   -----------------------------------------------------------------


--------------------------------------------------------------------
Machine Idioms
--------------------------------------------------------------------
      Description                               Optimized?
--------------------------------------------------------------------
Time : ss204 (          1.8 ) vs ss200 (          1.7 )     nostatistics
--------------------------------------------------------------------

      --------------------------------------------------------------
      ss204 ii := pp rem 4 ;
      -- natural integer REM, REM 4
      --------------------------------------------------------------
      ss200 ii := pp - ( ( pp / 4 ) * 4 ) ;
      -- expression comparable to MOD 4
      -- Optimization      :  machine idiom - AND masking operation
   -----------------------------------------------------------------


--------------------------------------------------------------------
Machine Idioms
--------------------------------------------------------------------
      Description                               Optimized?
--------------------------------------------------------------------
Time : ss207 (          0.5 ) vs ss208 (          0.7 )     yes
--------------------------------------------------------------------

      --------------------------------------------------------------
      If times are the same, system is NOT using special idioms
      for zero compare.

      ss207 IF ll < 0 THEN die ;  END IF ;
      -- machine idiom - does load set condition codes?
      -- Relational test, compare integer variable against 0
      --------------------------------------------------------------
      ss208 IF ll > 100 THEN die ;  END IF ;
      -- relational expression, integer / non-zero literal comparison.
      -- machine idiom -comparison with literal: compare immediate
   -----------------------------------------------------------------
```

```
-----------------------------------------------------------------------
Machine Idioms
-----------------------------------------------------------------------
        Description                                    Optimized?
-----------------------------------------------------------------------
Time : ss323 (          4.2 ) vs ss324 (          4.2 )      yes
-----------------------------------------------------------------------

        If times are the same, system is NOT using special idioms
        for zero compare.

        ss323 IF yy <= 0.0 THEN die ;  END IF ;
        -- floating point compare against zero
        -- machine idiom - load set condition codes against zero
        ---------------------------------------------------------------
        ss324 IF yy <= 0.1 THEN die ;  END IF ;
        -- floating point literal comparison against non-zero
        ---------------------------------------------------------------


-----------------------------------------------------------------------
Machine Idioms
-----------------------------------------------------------------------
        Description                                    Optimized?
-----------------------------------------------------------------------
Time : ss215 (          1.6 ) vs ss11 (          0.8 )      nostatistics
-----------------------------------------------------------------------

        ss215 ra.i := rb.i ;  ra.j := rb.j ;
        -- machine idiom, block move?  Copy two consecutively allocated
        -- fields from one instance of a record type to another.
        ---------------------------------------------------------------
        ss11  kk := 11 ;
        ---------------------------------------------------------------


-----------------------------------------------------------------------
Machine Idioms
-----------------------------------------------------------------------
        Description                                    Optimized?
-----------------------------------------------------------------------
Time : ss503 (          1.6 ) vs ss0 (          0.0 )      nostatistics
-----------------------------------------------------------------------

        ss503 ii := ii + 1 ;  ii := ii - 1 ;
        -- Increment and decrement same integer scalar -- could be noop
        ---------------------------------------------------------------
        ss0   NULL  ;
        ---------------------------------------------------------------


-----------------------------------------------------------------------
Machine Idioms
-----------------------------------------------------------------------
        Description                                    Optimized?
-----------------------------------------------------------------------
Time : ss205 (          0.8 ) vs ss206 (          0.8 )      yes
-----------------------------------------------------------------------

        ss205 IF 11 - mm > 0  THEN die ;  END IF ;
           -- Subtract two integers and compare result to 0
           -- arithmetic expression sets condition codes to
           -- reflect comparison against 0.  If so, no need
           -- for explicit compare
        ---------------------------------------------------------------
        ss206 IF 11 > mm  THEN die ;  END IF ;
           -- Directly compare two integers.  Compare with ss205.
           -- arithmetic expression sets condition codes to
```

```
            -- reflect comparison against 0.  If so, no need
            -- for explicit compare
------------------------------------------------------------------------

------------------------------------------------------------------------
Jump Tracing
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss250 (         0.0 ) vs ss0 (         0.0 )     yes
------------------------------------------------------------------------

        ss250 LOOP   EXIT ;  END LOOP ;  -- should be a noop
------------------------------------------------------------------------
        ss0   NULL ;
    --------------------------------------------------------------------


------------------------------------------------------------------------
Jump Tracing
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss619 (         0.4 ) vs ss0 (         0.0 )     nostatistics
------------------------------------------------------------------------

        ss619 <<10>> GOTO 11 ;  <<12>> GOTO 13 ;  <<14>> GOTO 15 ;
              <<16>> GOTO 17 ;  <<11>> GOTO 12 ;  <<13>> GOTO 14 ;
              <<15>> GOTO 16 ;  <<17>> null ;
        -- can be jump traced into a null.  6 "GOTO" statements which jump
        -- to another "GOTO" statement. Statements are not in order.
        -- Test for jump tracing optimization.
------------------------------------------------------------------------
        ss0   NULL ;
    --------------------------------------------------------------------


------------------------------------------------------------------------
Jump Tracing
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss620 (         0.0 ) vs ss0 (         0.0 )     yes
------------------------------------------------------------------------

        ss620 <<m0>> GOTO m1 ;  <<m1>> GOTO m2 ;  <<m2>> GOTO m3 ;
              <<m3>> GOTO m4 ;  <<m4>> GOTO m5 ;  <<m5>> GOTO m6 ;
              <<m6>> GOTO m7 ;  <<m7>> null ;
        -- A peephole optimizer which omits unconditional branch to
        -- the next instruction would suffice to optimize this
        -- 6 "GOTO" statements which branch to next statement.
        -- This is a simpler test for jump tracing that ss619.
        -- A peephole optimizer which omits a branch to the next
        -- instruction would suffice to optimize this problem.
------------------------------------------------------------------------
        ss0   NULL ;
    --------------------------------------------------------------------


------------------------------------------------------------------------
Jump Tracing
------------------------------------------------------------------------
        Description                                    Optimized?
------------------------------------------------------------------------
Time : ss26 (         0.3 ) vs ss0 (         0.0 )     nostatistics
------------------------------------------------------------------------

        ss26 GOTO 12;  <<11>> die;  GOTO 11;  <<12>> NULL;
```

```
              -- Language feature test, GOTO.
      ---------------------------------------------------------------
      ss0  NULL ;
    -------------------------------------------------------------


  -----------------------------------------------------------------------
  Jump Tracing
  -----------------------------------------------------------------------
        Description                              Optimized?
  -----------------------------------------------------------------------
  Time : ss261 (          0.0 ) vs ss0 (          0.0 )       yes
  -----------------------------------------------------------------------
        ---------------------------------------------------------------
        ss261 GOTO label;  <<label>> NULL;
           -- omittable code, either by flow analysis or peephole
        ---------------------------------------------------------------
        ss0  NULL ;
  -----------------------------------------------------------------------


  -----------------------------------------------------------------------
  Loop Fusion
  -----------------------------------------------------------------------
        Description                              Optimized?
  -----------------------------------------------------------------------
  Time : ss180 (          8.9 ) vs ss181 (          7.9 )      nostatistics
  -----------------------------------------------------------------------
        ---------------------------------------------------------------
        ss180 FOR i IN 1..5
              LOOP
                  i1 ( i ) := i ;
              END LOOP ;

              FOR i IN 1..5
              LOOP
                  e1 ( i ) := 1.0 ;
              END LOOP ;
           -- Problem has two separate FOR loops which can be fused.
        ---------------------------------------------------------------
        ss181 FOR i IN 1..5
              LOOP
                  i1 ( i ) := i ;
                  e1 ( i ) := 1.0 ;
              END LOOP ;
           -- Problem has one loop fused by hand.  Compare with ss180.
  -----------------------------------------------------------------------


  -----------------------------------------------------------------------
  Loop Interchange
  -----------------------------------------------------------------------
        Description                              Optimized?
  -----------------------------------------------------------------------
  Time : ss749 (        912.8 ) vs ss750 (        893.3 )      nostatistics
  -----------------------------------------------------------------------
        ---------------------------------------------------------------
        ss749 FOR j IN int'(1)..int'(10)
              LOOP
                  FOR i IN  int'(1)..int'(10)
                  LOOP                -- el(j) is an invariant in this LOOP
                      matrix ( j, i ) := e2 ( j, i ) + el ( j ) ** 2 ;
                  END LOOP;
              END LOOP;
           -- Optimization test FOR invariant LOOP code motion.  This
           -- example contains an invariant expression in an inner LOOP.
        ---------------------------------------------------------------
        ss750 FOR i IN   int'(1)..int'(10)
```

```
              LOOP
                 FOR j IN int'(1)..int'(10)
                 LOOP              -- el(j) is not invariant in this LOOP
                    matrix ( i, j ) := e2 ( i, j ) + el ( j ) ** 2 ;
                 END LOOP;
              END LOOP;
      -- Test FOR LOOP interchange optimization.
```

Loop Unrolling, Test Elimination

Description	Optimized?

Time : ss541 (102.3) vs ss542x (94.8)   nostatistics

```
         ss541 FOR i IN 1..ei * 10
               LOOP
                  IF i = 1
                  THEN
                     el ( i ) := ( 1.0 + el ( i ) ) / 2.0 ;
                  ELSE
                     el ( i ) := ( el ( i - 1 ) + el ( i ) ) / 2.0 ;
                  END IF;
               END LOOP;
         -- unrolling, test elimination. This has variable upper bound.
```

```
         ss542x el ( 1 ) := ( 1.0 + el ( 1 ) ) / 2.0 ;
               IF ei >= 1
               THEN
                  FOR i in 2..ei*10
                  LOOP
                     el ( i ) := ( el ( i - 1 ) + el ( i ) ) / 2.0 ;
                  END LOOP;
               END IF;
         -- unrolled version of ss541
```

Loop Unrolling, Test Elimination

Description	Optimized?

Time : ss542 (100.9) vs ss542x (94.8)   no

```
         ss542 FOR i in 1..10
               LOOP
                  IF i = 1
                  THEN
                     el ( i ) := ( 1.0 + el ( i ) ) / 2.0 ;
                  ELSE
                     el ( i ) := ( el ( i - 1 ) + el ( i ) ) / 2.0 ;
                  END IF;
               END LOOP;
         -- unrolling, test elimination.  ss541 with literal upper bound.
```

```
         ss542x el ( 1 ) := ( 1.0 + el ( 1 ) ) / 2.0 ;
               IF ei >= 1
               THEN
                  FOR i in 2..ei * 10
                  LOOP
                     el ( i ) := ( el ( i - 1 ) + el ( i ) ) / 2.0 ;
                  END LOOP;
```

```
              END IF;
        -- unrolled version of ss541
-------------------------------------------------------------------------


-------------------------------------------------------------------------
Loop Unrolling
-------------------------------------------------------------------------
        Description                                  Optimized?
-------------------------------------------------------------------------
Time : ss105 (        27.4 ) vs ss642 (        22.1 )    nostatistics
-------------------------------------------------------------------------

        -------------------------------------------------------------------
        -------------------------------------------------------------------
        ss105 FOR i IN 1..10 LOOP proc0 ;  END LOOP ;
        -- FOR LOOP, containing procedure call.
        -- Optimization      :  LOOP unrolling
        -------------------------------------------------------------------
        ss642 proc0 ;   .. proc0 ;       -- 10 calls on proc0
        -- Sequence of procedure calls.  Timing consistency check.
        -------------------------------------------------------------------


-------------------------------------------------------------------------
Loop Unrolling
-------------------------------------------------------------------------
Test        Execution   Bar                              Similar
Name        Time        Chart                            Groups
-------------------------------------------------------------------------
ss238       0.78    *************                    |
ss3         0.78    *************                    |
ss17        1.41    ***********************           |
ss57        1.42    ***********************           |
ss240       2.07    ************************************      |
-------------------------------------------------------------------------
                       Individual Test Descriptions
-------------------------------------------------------------------------

    -------------------------------------------------------------------
    ss3  xx := yy ;
    -- Assignment of two floating point variables, library scope.
    -------------------------------------------------------------------
    ss17 el ( ei ) := one ;
    -- assignment to one dimensional array of real.
    -------------------------------------------------------------------
    ss57 el ( i ) := one ;   -- i is LOOP index
    -- Test subscript computation using FOR LOOP index.
    -------------------------------------------------------------------
    ss238 FOR i IN 1..1 LOOP    el ( i ) := one ;    END LOOP ;
    -- can unroll LOOP into single assignment statement
    -- simple example amenable to LOOP unrolling
    -------------------------------------------------------------------
    ss240 FOR i IN 1..2 LOOP    el ( i ) := one ;    END LOOP ;
    -- simple example amenable to LOOP unrolling
    -------------------------------------------------------------------
-------------------------------------------------------------------------


-------------------------------------------------------------------------
Loop Flattening : 2 Dimensional Arrays Of Real
-------------------------------------------------------------------------
Test        Execution   Bar                              Similar
Name        Time        Chart                            Groups
-------------------------------------------------------------------------
ss18        3.96    *                                |
ss405       156.40  ********************************* |
-------------------------------------------------------------------------
                       Individual Test Descriptions
-------------------------------------------------------------------------
```

If time to execute ss405 is less than 100 times the time to execute
ss18, then the compilation system is treating subscript calculations
using for loop indexes better than general usage. May be using
strength reduction, register allocation, or other techniques including
loop flattening. Flattening is the merging of the two nested loops
into one larger loop.

```
e2 : ARRAY ( int'(1)..int'(10) ,int'(1)..int'(10) ) OF real
              := ( int'(1)..int'(10) =>( int'(1)..int'(10) =>1.0));
ei, ej, ek : int := 1;
```

```
ss18  e2 ( ei, ej ) := one ;
   -- assignment to two dimensional array of real. Checking.
```

```
ss405 FOR i IN 1 .. 10 LOOP
         FOR j IN 1 .. 10 LOOP
            e2 ( int ( i ), int ( j )  ) := one ;
         END LOOP ;
      END LOOP ;
   -- nested FOR loop to access a 2D array -- loops could be flattened
```

Loop Invariant Motion

Description	Optimized?
Time : ss212 (9.9) vs ss3 (0.8)	no

```
      ss212 FOR i IN 1..10 LOOP   xx := yy ;   END LOOP ;
      -- example where invariant motion is possible
```
```
      ss3   xx := yy ;
```

Loop Invariant Motion

Description	Optimized?
Time : ss429 (3.2) vs ss430 (3.2)	yes

```
      FUNCTION a1 ( i : int ) RETURN int IS
         ca1 : CONSTANT ARRAY ( int'(0)..int'(2) ) OF int := (0, 1, 2) ;
      BEGIN
         RETURN ca1 ( i ) ;
      END a1 ;
```
```
      ss429 ii := a1 ( ei ) ;
      -- Is constant static array promoted to outer level?
```
```
      ca2 : CONSTANT ARRAY ( int'(0)..int'(2) ) OF int := (0, 1, 2) ;
      FUNCTION a2 ( i : int ) RETURN int IS
      BEGIN
         RETURN ca2 ( i ) ;
      END a2 ;
```
```
      ss430 ii := a2 ( ei ) ; --non-local constant array
      -- Is constant static array promoted to outer level?
```

Loop Invariant Motion

```
------------------------------------------------------------------
        Description                              Optimized?
------------------------------------------------------------------
Time : ss536 (      283.1 ) vs ss535 (       91.7 )   nostatistics
------------------------------------------------------------------

        ss536 FOR l IN 1..mm LOOP
                xx := 0.0 ;
                FOR k IN el'RANGE LOOP
                    xx := xx + el ( k ) ** 2 ;
           .    END LOOP ;
              END LOOP ;   -- xx is computed from invariants in 'l' loop
        -- very smart optimizer can do inner loop once
        ----------------------------------------------------------

        ss535 xx := 0.0 ;
              FOR k IN el'RANGE LOOP
                  xx := xx + el ( k ) ** 2 ;
              END LOOP ;   -- sample to embed in code for ss536
        ----------------------------------------------------------
```

```
------------------------------------------------------------------
Loop Invariant Motion
------------------------------------------------------------------
        Description                              Optimized?
------------------------------------------------------------------
Time : ss752 (        9.9 ) vs ss11 (       0.8 )   nostatistics
------------------------------------------------------------------

        ss752 FOR i IN 1..10 LOOP  ii := jj ;  END LOOP ;
        -- could be optimized into an assignment statement, ss11
        ----------------------------------------------------------
        ss11  kk := ll ;  -- Library scope integer assignment.
        ----------------------------------------------------------
```

```
------------------------------------------------------------------
FOR LOOP with NULL body
------------------------------------------------------------------
        Description                              Optimized?
------------------------------------------------------------------
Time : ss106 (        5.8 ) vs ss0 (       0.0 )   nostatistics
------------------------------------------------------------------

        ss106 FOR i IN 1..10
              LOOP
                  NULL ;
              END LOOP ;   -- noop
        -- FOR loop with null body, could be noop.
        ----------------------------------------------------------
        ss0   NULL ;
        ----------------------------------------------------------
```

```
------------------------------------------------------------------
Test Merging
------------------------------------------------------------------
        Description                              Optimized?
------------------------------------------------------------------
Time : ss178 (        2.3 ) vs ss179 (       2.9 )   yes
------------------------------------------------------------------

        ss178 IF ll > mm
              THEN   ii := 0 ;
              ELSE   ii := 1 ;
              END IF ;
              IF ll > mm
```

```
                THEN   xx := 0.0 ;
                ELSE   xx := 1.0 ;
                END IF ;
        -- Problem has tests which may be merged.
        ------------------------------------------------------------
        ss179 IF ll > mm
                THEN   ii := 0 ;
                       xx := 0.0 ;
                ELSE   ii := 1 ;
                       xx := 1.0 ;
                END IF ;
        -- Problem has tests in ss178 merged by hand.
   -----------------------------------------------------------------


   -----------------------------------------------------------------
Test Merging
   -----------------------------------------------------------------
        Description                                    Optimized?
   -----------------------------------------------------------------
Time : ss440 (        11.0 ) vs ss441 (        9.2 )    nostatistics
   -----------------------------------------------------------------
        ------------------------------------------------------------
        ss440 FOR i IN 1..2
                LOOP
                    IF ii = 1 THEN il ( ll ) := ll ;  END IF ;
                    IF ii = 1 THEN el ( ei ) := one ; END IF ;
                    IF ii/= 1 THEN proc0 ; END IF ;
                    ii := 1 - ii ;
                END LOOP ;
        -- test merging.  Several IF's can be merged.
        ------------------------------------------------------------
        ss441 FOR i IN 1..2
                LOOP
                    IF ii = 1
                    THEN
                        il ( ll ) := ll ;
                        el ( ei ) := one ;
                    ELSE
                        proc0;
                    END IF ;
                    ii := 1 - ii ;
                END LOOP ;
        -- This version has merged tests, compare with ss440
   -----------------------------------------------------------------


   -----------------------------------------------------------------
Respect for Parentheses Test
   -----------------------------------------------------------------
Test       Execution    Bar                             Similar
Name       Time         Chart                           Groups
   -----------------------------------------------------------------
ss69          6.28      ****************************       |
ss70          7.85      *********************************  |
   -----------------------------------------------------------------
                     Individual Test Descriptions
   -----------------------------------------------------------------
IF ss69 = ss70 THEN parentheses are NOT respected.
   --------------------------------------------------------------
ss69 xx := 1.0 - yy;
-- This is a folded version of ss70.
   --------------------------------------------------------------
ss70 xx := ( 0.5 - yy ) + 0.5;
-- This might be improperly folded into ss69.
   -----------------------------------------------------------------
```

```
------------------------------------------------------------------------
Superfluous Parentheses
------------------------------------------------------------------------
        Description                                  Optimized?
------------------------------------------  ----------------------------
Time : ss389 (          0.8 ) vs ss3 (          0.8 )     yes
------------------------------------------------------------------------

        ss389 xx := ( yy ) ;
        -- Do superfluous parentheses produce code?
        ------------------------------------------------------------
        ss3   xx := yy ;
        -- Assignment of two floating point variables, library scope.
        ------------------------------------------------------------


------------------------------------------------------------------------
Superfluous Parentheses
------------------------------------------------------------------------
        Description                                  Optimized?
------------------------------------------  ----------------------------
Time : ss391 (          8.0 ) vs ss390 (          8.0 )     yes
------------------------------------------------------------------------

        ss391 xx := ( one + yy ) + zz ;
        -- Add 3 float variables, parentheses around first two.
        ------------------------------------------------------------
        ss390 xx := one + yy + zz ;
        -- Add 3 float variables
        ------------------------------------------------------------


------------------------------------------------------------------------
Superfluous Parentheses
------------------------------------------------------------------------
        Description                                  Optimized?
------------------------------------------  ----------------------------
Time : ss392 (          8.0 ) vs ss390 (          8.0 )     yes
------------------------------------------------------------------------

        ss392 xx := one + ( yy + zz ) ;
        -- Add 3 float variables, parentheses around last two.
        ------------------------------------------------------------
        ss390 xx := one + yy + zz ;
        -- Add 3 float variables
        ------------------------------------------------------------


------------------------------------------------------------------------
Superfluous Parentheses
------------------------------------------------------------------------
        Description                                  Optimized?
------------------------------------------  ----------------------------
Time : ss393 (          0.8 ) vs ss11 (          0.8 )     yes
------------------------------------------------------------------------

        ss393 ii := ( mm ) ;
        ------------------------------------------------------------
        ss11  kk := ll ;
        -- Library scope integer assignment.
        ------------------------------------------------------------


------------------------------------------------------------------------
Superfluous Parentheses
------------------------------------------------------------------------
        Description                                  Optimized?
------------------------------------------  ----------------------------
Time : ss395 (          1.3 ) vs ss394 (          1.3 )     yes
```

```
------------------------------------------------------------------
      ss395 ii := ( ei + ej ) + ll ;
      -- Add 3 integer variables, parentheses around first two.
------------------------------------------------------------------
      ss394 ii := ei + ej + ll ;
      -- Add 3 integer variables
  ------------------------------------------------------------------


------------------------------------------------------------------
Superfluous Parentheses
------------------------------------------------------------------
      Description                             Optimized?
------------------------------------------------------------------
Time : ss396 (         1.3 ) vs ss394 (        1.3 )     yes
------------------------------------------------------------------

  ------------------------------------------------------------------
      ss396 ii := ei + ( ej + ll ) ;
      -- Add 3 integer variables, parentheses around last two.
------------------------------------------------------------------
      ss394 ii := ei + ej + ll ;
      -- Add 3 integer variables
------------------------------------------------------------------


------------------------------------------------------------------
Order Of Evaluation & Register Allocation Test For Parameters
------------------------------------------------------------------
      Description                             Optimized?
------------------------------------------------------------------
Time : ss546 (        23.0 ) vs ss547 (       21.6 )    nostatistics
------------------------------------------------------------------

  ------------------------------------------------------------------
      ss546 ii := max ( 1, max ( 2, max ( 3, max ( 4, max ( 5, max (
         6, max ( 7, max ( 8, max ( 9, 10 ) ) ) ) ) ) ) ) ) ;
      -- non left-to-right order of evaluation can reduce register
      -- save/restore activity.  Call on two parameter function, with
      -- left actual parameter a literal and right a further function call.

      -- Nested 8 levels.  Integer function (max).  A strict left to right
      -- order of evaluation will result in unnecessary storing and loading
      ------------------------------------------------------------------
      ss547 ii := max ( max ( max ( max ( max ( max ( max ( max ( max
         ( 10, 9 ),   8 ),   7 ),   6 ),   5 ),   4 ),   3 ),   2 ) , 1 ) ;
      -- Analogous to ss546 with calls nested on first parameter.
      -- A left to right order of evaluation is best here.  Good
      -- compiler will do both ss546 and ss547 is about the same time.
  ------------------------------------------------------------------


------------------------------------------------------------------
Order Of Evaluation & Register Allocation Test For Parameters
------------------------------------------------------------------
      Description                             Optimized?
------------------------------------------------------------------
Time : ss548 (        80.1 ) vs ss549 (       84.0 )     yes
------------------------------------------------------------------

  ------------------------------------------------------------------
      ss548 xx := max ( 1.0, max ( 2.0, max ( 3.0, max ( 4.0, max ( 5.0,
         max ( 6.0, max ( 7.0, max ( 8.0, max ( 9.0, 10.0 )))))))))) ;
      -- non left-to-right order of evaluation can reduce
      -- register save/restore activity
      ------------------------------------------------------------------
      ss549 xx := max ( max ( max ( max ( max ( max ( max ( max ( max
         ( 10.0, 9.0), 8.0), 7.0), 6.0), 5.0), 4.0), 3.0), 2.0), 1.0) ;
      -- cf ss548,  for optimizing compilers,  should be about same
      ------------------------------------------------------------------
```

```
------------------------------------------------------------------
Order Of Evaluation & Register Allocation Test For Parameters
------------------------------------------------------------------
        Description                              Optimized?
------------------------------------------------------------------
Time : ss550 (        15.5 ) vs ss551 (        15.2 )    nostatistics
------------------------------------------------------------------
        ----------------------------------------------------------
        i1  : int RENAMES global.i1 ( 1 ) ;
        ::                              ::
        i10 : int RENAMES global.i1 ( 10 ) ;
        ----------------------------------------------------------
        ss550 ii := ( i1 + ( i2 + ( i3 + ( i4 + ( i5 + ( i6 + ( i7 +
                  ( i8 + ( i9 +i10 )))))))) ;
        -- Integer addition with parameters nested on second operand.
        -- A left-to-right order of evaluation may generate unnecessary
        -- stores and reloads.
        ----------------------------------------------------------
        ss551 ii := (((((((( i10 + i9 ) + i8 ) + i7 ) + i6 ) +
                  i5 ) + i4 ) + i3 ) + i2 ) + i1 ) ;
        -- Integer addition with parameters nested on first operand.
        -- A left-to-right order of evaluation is best.
        ----------------------------------------------------------


------------------------------------------------------------------
Register Allocation With Call On External Procedure
------------------------------------------------------------------
Test        Execution    Bar                             Similar
Name        Time         Chart                           Groups
------------------------------------------------------------------
ss442       106.30       *********************************  |
ss443       106.30       *********************************  |
------------------------------------------------------------------
                    Individual Test Descriptions
------------------------------------------------------------------
IF ss443 << ss442 THEN opt := YES
------------------------------------------------------------------
ss442 xx := 0.0 ;
      FOR i IN el'RANGE
      LOOP
          xx := xx + el ( i ) ;
          IF el ( i ) > 2.0        -- never true
          THEN
              xx := el ( i ) ;
              die ;                -- die is a  global procedure
          END IF ;
      END LOOP ;
      -- register allocation - with call on external procedure,
      -- compiler cannot allocate "xx" to register within FOR LOOP.
------------------------------------------------------------------
ss443 xx := 0.0 ;
      FOR i IN el'RANGE
      LOOP
          xx := xx + el ( i ) ;
          IF el ( i ) > 2.0        -- never true
          THEN
              xx := el ( i ) ;
          END IF ;
      END LOOP ;
      -- register allocation - no call on "die" so
      -- xx can be allocated to register
------------------------------------------------------------------


------------------------------------------------------------------
```

```
Register Allocation
------------------------------------------------------------------
      Description                                Optimized?
------------------------------------------------------------------
Time : ss262 (        18.4 ) vs ss263 (        11.8 )    nostatistics
------------------------------------------------------------------

      ----------------------------------  ------------------------------
   ss262 xx := 0.1 - yy ;
         IF xx < 0.0 THEN   xx := -xx ;        END IF ;
         IF xx > 1.0 THEN   xx := 1.0 / xx ;   END IF ;
         -- only 1 register store for xx is required if compiler
         -- tracks registers.  See time for ss263
         -- example where good register usage would show up.
         -- Floating point variable is used in several consecutive
         -- IF statements.
      ------------------------------------------------------------
   ss263 xx := abs ( 0.1 - yy ) ;
         IF xx > 1.0 THEN   xx := 1.0 / xx ;  END IF ;
         -- example where good register usage would show up.
         -- Variable used in 2 consecutive statements.
------------------------------------------------------------------


------------------------------------------------------------------
Register Allocation
------------------------------------------------------------------
      Description                                Optimized?
------------------------------------------------------------------
Time : ss264 (         2.6 ) vs ss266 (         1.6 )    no
------------------------------------------------------------------

      ------------------------------------------------------------
   ss264 ii := jj ;  IF ii < 0 THEN   ii := -ii ;   END IF ;
         -- example where good register usage would show up.
         -- Integer variable stored in one statement is referenced
         -- in relational test and in the THEN clause of the statement.
      ------------------------------------------------------------
   ss266 ii := abs ( jj );
         -- integer abs
------------------------------------------------------------------


------------------------------------------------------------------
Register Allocation
------------------------------------------------------------------
      Description                                Optimized?
------------------------------------------------------------------
Time : ss265 (         2.0 ) vs ss266 (         1.6 )    no
------------------------------------------------------------------

      ------------------------------------------------------------
   ss265 ii := jj ;  ii := abs ( ii ) ;
         -- example where good register usage would show up.
         -- Integer variable stored in one statement is referenced
         -- in the next statement.
      ------------------------------------------------------------
   ss266 ii := abs ( jj );
         -- integer abs
------------------------------------------------------------------


------------------------------------------------------------------
Relational Expression OR vs OR ELSE
------------------------------------------------------------------
      Description                                Optimized?
------------------------------------------------------------------
Time : ss224 (         2.3 ) vs ss223 (         2.2 )    no
------------------------------------------------------------------

      ------------------------------------------------------------
      ------------------------------------------------------------
```

An optimizing compiler could determine that the relations 'mm=139'
and 'mm > 1000' could not have side effects (other than perhaps
raising an exception which would be permissible to ignore) and
so it could validly treat the 'OR' operator as an 'OR ELSE'.

```
---------------------------------------------------------------
ss223 IF mm = 11 OR mm = 139 OR mm > 1000
        THEN
            die ;
        END IF ;
-- relational expression example, OR
---------------------------------------------------------------
ss224 IF mm = 11 OR ELSE mm = 139 OR ELSE mm > 1000
        THEN
            die ;
        END IF ;
-- relational expression example, OR ELSE.
---------------------------------------------------------------
```

```
---------------------------------------------------------------
IF Statement - Integer, Float - AND vs AND THEN
---------------------------------------------------------------
```

Description	Optimized?
Time : ss88 (5.5) vs ss89 (1.2)	nostatistics

```
---------------------------------------------------------------

---------------------------------------------------------------
ss88  IF  11 > mm  AND  yy > zz
        THEN
            Die ;
        END IF; -- False
-- IF statement, integer and floating relation (false) "AND"
---------------------------------------------------------------
ss89  IF  11 > mm  AND THEN  yy > zz
        THEN
            Die ;
        END IF ; -- False
-- IF statement, integer and floating relation (false) "AND THEN"
---------------------------------------------------------------
```

```
---------------------------------------------------------------
IF Statement - Integer Relations, Simplifications
---------------------------------------------------------------
```

Description	Optimized?
Time : ss228 (1.3) vs ss229 (1.3)	yes

```
---------------------------------------------------------------

---------------------------------------------------------------
```

Because the possibility of side effects can be eliminated at
compile time, an optimizing compiler could treat these statements
the same.

```
ss228 IF 11 > mm OR bool
        THEN
            die ;
        END IF ;
-- boolean expression, integer relation OR boolean variable
---------------------------------------------------------------
ss229 IF 11 > mm OR ELSE bool
        THEN
            die ;
        END IF ;
-- boolean expression, integer relation OR ELSE boolean variable.
---------------------------------------------------------------
```

```
----------------------------------------------------------------------
IF Statement - Integer Relations, Simplifications
----------------------------------------------------------------------
      Description                                Optimized?
----------------------------------------------------------------------
Time : ss231 (          1.5 ) vs ss84 (          0.8 )    nostatistics
----------------------------------------------------------------------

----------------------------------------------------------------- -
      ss231 IF ll > mm OR ELSE False
                      OR ELSE False
            THEN
               die ;
            END IF ;
      -- foldable boolean expression "OR ELSE False OR ELSE False"
      -----------------------------------------------------------------
      ss84 IF ll > mm
            THEN
               die ;
            END IF ;  -- False
      -- IF statement, integer relation (False), no ELSE clause
    ------------------------------------------------------------------
----------------------------------------------------------------------


----------------------------------------------------------------------
Strength Reduction
----------------------------------------------------------------------
      Description                                Optimized?
----------------------------------------------------------------------
Time : ss213 (       233.3 ) vs ss422 (       47.6 )    no
----------------------------------------------------------------------

    ------------------------------------------------------------------
    ------------------------------------------------------------------
      ss213 ii := 0 ;
            FOR i in 1..10
            LOOP
               ii := ii + il ( i ) * ( -1 ) ** i ;
            END LOOP ;
      -----------------------------------------------------------------
      ss422 kk := +1 ;
            ii := 0 ;
            FOR i in 1..10
            LOOP
               ii := ii + il ( i) * kk ;   kk := -kk ;
            END LOOP ;
      -- Strength reduction, by hand, of ss213.  Reduces an
      -- exponential by FOR LOOP index " ( -1) ** i"
      -----------------------------------------------------------------


----------------------------------------------------------------------
Strength Reduction
----------------------------------------------------------------------
      Description                                Optimized?
----------------------------------------------------------------------
Time : ss423 (       16.7 ) vs ss424 (       20.6 )    yes
----------------------------------------------------------------------

      ss423 isum := 0 ;
            FOR i in 1..ten
            LOOP
               ii := i * 2 ;
               isum := isum + ii ;
            END LOOP ;
      -- Has multiply by FOR LOOP index
      -----------------------------------------------------------------
      ss424 ii := 0 ;
```

```
                      isum := 0 ;
                      FOR i in 1..ten
                      LOOP
                          ii := ii + 2 ;  isum := isum + ii ;
                      END LOOP ;
              -- Hand reduced form of ss423,
              -- with multiply by FOR LOOP index reduced to add.
          ----------------------------------------------------------------
```

```
------------------------------------------------------------------------
Strength Reduction
------------------------------------------------------------------------
          Description                                  Optimized?
------------------------------------------------------------------------
Time : ss425 (          33.9 ) vs ss426 (          36.6 )      yes
------------------------------------------------------------------------

              ----------------------------------------------------------
              ss425 analogous to ss423 with WHILE LOOP ;
              -- Multiply by induction variable
              -- which is not a FOR LOOP index.
              ------------------------------------------  ----------------
              ss426 analogous to ss424 with WHILE LOOP ;
              -- Hand reduced form of ss425
              -- with multiply reduced to add. Induction variable
              -- is not a FOR LOOP index.
              ----------------------------------------------------------
```

```
------------------------------------------------------------------------
Strength Reduction
------------------------------------------------------------------------
          Description                                  Optimized?
------------------------------------------------------------------------
Time : ss15 (          6.8 ) vs ss5 (          6.8 )      yes
------------------------------------------------------------------------

              ----------------------------------------------------------
              ss15 xx := yy ** 2 ;
              -- (float) ** 2 which can be treated as (float) * (float).
              ----------------------------------------------------------
              ss5  xx := yy * zz ;
              ----------------------------------------------------------
```

```
------------------------------------------------------------------------
Strength Reduction
------------------------------------------------------------------------
          Description                                  Optimized?
------------------------------------------------------------------------
Time : ss188 (          2.7 ) vs ss202 (          3.0 )      yes
------------------------------------------------------------------------

              ----------------------------------------------------------
              ss188 ii := ll ** 2 ; -- could be strength reduced to ll * ll
              ----------------------------------------------------------
              ss202 ii := ll * mm ;
              -- integer multiplication
              ----------------------------------------------------------
```

```
------------------------------------------------------------------------
Strength Reduction
------------------------------------------------------------------------
          Description                                  Optimized?
------------------------------------------------------------------------
Time : ss279 (          2.8 ) vs ss273 (          2.8 )      yes
------------------------------------------------------------------------

              ----------------------------------------------------------
              ss279 li := lj ** 2 ;
              -- bigint type **2
```

```
-----------------------------------------------------------------------
ss273 li := lj * lk ;
-- bigint type multiplication
-----------------------------------------------------------------------


-----------------------------------------------------------------------
Test Swapping
-----------------------------------------------------------------------
      Description                            Optimized?
-----------------------------------------------------------------------
Time : ss438 (        155.8 ) vs ss439 (        136.5 )    nostatistics
-----------------------------------------------------------------------

      -----------------------------------------------------------------
      ss438 FOR i IN el'range
            LOOP
                IF bool
                THEN
                    el ( i ) := xel ( i ) + zero * yel ( i ) ;
                    xel ( i ) := one * yel ( i ) ;
                ELSE
                    el ( i ) := xel ( i ) - zero * yel ( i ) ;
                END IF ;
            END LOOP ;
      -- test swapping. FOR LOOP with embedded IF statement
      -- with LOOP invariant expressions in relation and in the
      -- conditional statements.  "IF" can be moved out of FOR
      -- LOOP as done by hand in ss439.
      -----------------------------------------------------------------
      ss439 IF bool
            THEN
                FOR i IN el'range
                LOOP
                    el ( i ) := xel ( i ) + zero * yel ( i ) ;
                    xel ( i ) := one * yel ( i ) ;
                END LOOP ;
            ELSE
                FOR i in el'range
                LOOP
                    el ( i ) := xel ( i ) - zero * yel ( i ) ;
                END LOOP ;
            END IF ;
      -- test swapping.  Hand optimized version of ss438.
      -----------------------------------------------------------------
```

References

[VADScross-a] VADScross Verdix Ada Development System for Cross-Development Environ-
               ments, Version 6.05; VAX VMS ⇒ MC68020/30; User's Guide; Verdix Corpo-
               ration; January 16, 1991.

[VADScross-b] VADScross Verdix Ada Development System for Cross-Development Environ-
               ments, Version 6.05; VAX VMS ⇒ MC68020/30; Programmer's Guide; Verdix
               Corporation; January 16, 1991.

2.13 Precision

Question: What are the performance differences between single-precision and extended-precision numeric operations?

Summary: Assignment, arithmetic operations, and computations involving math-library functions all take longer to perform when computed with 9 or 15 digits of floating-point precision rather than with 6 digits.The performance degradation attributable to extended-precision arithmetic ranges from 11% for the exponentiation function to 100% for a simple assignment statement. In general, the percentage increase in execution time was larger for the single-statement tests than for composite tests such as the Whetstone benchmark (14%) and the PIWG B tests (19% increase with checking enabled, 23% increase with checking suppressed).

Discussion: Variables of an extended-precision type require more storage than their single-precision counterparts, so it is to be expected that there is a performance penalty associated with the use of such floating-point and fixed-point types. Users may be required to balance the need for greater precision in computations with the efficiency of those computations. Similarly, users may define integer types with various ranges that require, for example, 16 bits of storage to implement one type and 32 bits to implement another. A comparison of the performance of operations on these different sizes of objects may guide the user in specifying a required precision or a range of values that maximizes the efficiency and minimizes the storage overhead of the resultant code.

The AES, ACEC, and PIWG suites all contain tests to measure the performance of operations on fixed-point variables but, unfortunately, none of them contains a complementary set of tests to measure the performance of the same set of operations on fixed-point variables of greater or lesser precision. The ACEC has a complementary set of tests for different sizes of integer objects, but the implementation of the tests is flawed; see the second observation below for details.

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: The results from the three benchmark suites show that, in all cases, there is a performance price to pay for extended-precision floating-point computations.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0,Tests TI05 and TI05b.

The performance degradation attributable to extended-precision arithmetic ranges from 13% for exponentiation (to a power of two) to 70% for division. For addition and multiplication, the percentages are, respectively, 35% and 48%. The tests use the pre-defined types Short_Float (6 digits of precision) and Float (15 digits of precision). These types require 32 and 64 bits of storage, respectively.

I. Group I - Runtime Efficiency Tests

I.13. TI05

This test examines the efficiency of floating point computations, in particular, the average cpu time taken for performing a single operation of addition, multiplication, division and exponentiation for each of the predefined floating point types. The times measured include the overhead of an access to a simple variable.

For the addition operation a minimum and maximum time is given. The maximum time corresponds to the case where the greatest amount of shifting of the operands is required in order to align the decimal points so that the addition can be performed.

The minimum time given corresponds to the case where no addition is actually performed, because the operands are so widely differing in magnitude. Rather, the largest operand is selected as the result.

There is likely to be little difference in these maxima and minima if the addition is performed in hardware.

The exponentiation test is performed twice, once when all the real operands are raised to the power of 2 and then again when exponents are generated which are in the range that is safe for each real operand.

Exponentiation is defined in terms of repeated multiplication in Ada (and division for negative exponents) but an implementation may choose an alternative method of calculation (eg. by taking logs or via look-up tables).

Floating point type : short_float

```
        Addition (minimum time)              : 5.73us
        Addition (maximum time)              : 5.73us
        Multiplication                       : 5.85us
        Division                             : 6.29us
        Exponentiation (to power of 2)       : 16.4us
        Exponentiation (to any safe power)   : 14.8us
```

Floating point type : float

```
        Addition (minimum time)              : 7.76us
        Addition (maximum time)              : 7.76us
        Multiplication                       : 8.68us
        Division                             : 10.7us
        Exponentiation (to power of 2)       : 18.6us
        Exponentiation (to any safe power)   : 21.9us
```

I.14. TI05B

This test examines the efficiency of floating point computations, in particular, the average cpu time taken

```
        for performing a  single  evaluation  of  a  mathematical
        function for each of the predefined floating point types.
        The  mathematical  functions  evaluated are sines, square
        roots and natural logarithms. The times measured  include
        the overhead of an access to a simple variable.

        Test failed. Malfunction in Test Harness
        Exception in Unattended mode
```

ACEC Test Results:

Configuration 1, ACEC Release 2.0, Raw Output

The table below is a summary of ACEC raw test results; it is not produced directly by the ACEC analysis tools. (Apart from the tests that assign a value to an element of an array, none of these tests is processed by the Single-System Analysis (SSA) tool of the ACEC, so the process of gathering the data and computing the percentage degradation in speed is labor-intensive.) Test descriptions are symbolic representations of the Ada language feature being measured. The "Cnvrt" operation represents the conversion of a literal value to the appropriate type.

Whetstone benchmark results are reported in Kilo-Whetstone Instructions Per Second (KWIPS). All other results in this table are reported in microseconds. There are no subtraction tests in the suite, and there is no extended-precision multiplication test; these omissions appear to be an oversight on the part of the test developers. Unlike the AES tests, the ACEC tests did not use the predefined floating-point types; user-defined types with 6 and 9 digits of precision were used instead.

Table 3: ACEC Floating-Point Results

Test Description		6-Digit Precision	9-Digit Precision	Percent Degradation
Assignment:	$y = 1.0$	0.8	1.6	100.0%
Assignment:	$y = Cnvrt (1.0)$	0.8	1.6	100.0%
Assignment:	$y = x$	0.8	1.6	100.0%
Assignment:	$a(i) = x$	1.4	2.2	57.1%
Assignment:	$y = abs(x)$	4.0	6.5	62.5%
Addition:	$y = x + z$	6.3	9.0	42.8%
Subtraction:	$y = x - z$	N/A	N/A	
Multiplication:	$y = x * z$	6.8	N/A	
Division:	$y = x / z$	7.2	11.3	56.9%
Exponentiation:	$y = x ** 2$	6.8	9.7	42.6%
Trigonometric:	$y = sin(x)$	64.2	76.0	18.3%
Trigonometric:	$y = cos(x)$	71.6	94.8	18.4%
Trigonometric:	$y = exp(x)$	107.7	120.5	11.8%

Table 3: ACEC Floating-Point Results

Test Description		6-Digit Precision	9-Digit Precision	Percent Degradation
Trigonometric:	y = log(x)	137.9	153.9	11.6%
Trigonometric:	y = sqrt(x)	51.8	64.6	24.7%
Trigonometric:	y = arctan(x)	296.2	368.3	24.3%
Whetstone benchmark		914 KWIPS	805 KWIPS	13.5%

PIWG Test Results:

Configuration 1, PIWG 12/12/87 Release, Tests B000002 and B000003.

The PIWG B tests represent portions of an actual radar tracking application. The tests initia ⁻ ⁻ and then update a covariance matrix. When runtime checks are enabled, there is a 19% increase in the execution time of the 9-digit test over the 6-digit test. With checks suppressed, the difference is 23%. Like the ACEC tests, the PIWG tests used user-defined types with 6 and 9 digits of precision.

```
B000002  application program, tracker
TRACK WITH COVARIANCE MATRIX FLOAT 6 DIGITS

Time Required :  2.51790000000000E+01 Seconds for  10000 Repetitions

TRACK WITH COVARIANCE MATRIX FLOAT 6 DIGITS - SUPPRESS

Time Required :  1.99250000000000E+01 Seconds for  10000 Repetitions


B000003  application program, tracker
TRACK WITH COVARIANCE MATRIX - FLOAT 9 DIGITS

Time Required :  2.99110000000000E+01 Seconds for  10000 Repetitions

TRACK WITH COVARIANCE MATRIX - FLOAT 9 DIGITS  SUPPRESS

Time Required :  2.44450000000000E+01 Seconds for  10000 Repetitions
```

Observation 2: Analogous to the precision of floating-point types is the size of integer types. The test results below show the performance consequences of using different sizes of integer types.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, Raw Output.

The table below is a summary of ACEC raw test results; it is not produced directly by the ACEC analysis tools. Test descriptions are symbolic representations of the Ada language feature being measured. The "Cnvrt" operation represents the conversion of a literal value to the appropriate type.

Table 4: ACEC Integer Results

Test Description		6-Digit Precision	9-Digit Precision	Percent Degradation
Assignment:	y = 1	0.8	N/A	
Assignment:	y = Cnvrt (1)	0.8	N/A	
Assignment:	y = x	0.8	0.8	0.0%
Addition:	y = x + z	1.1	1.1	0.0%
Subtraction:	y = x - z	N/A	N/A	
Multiplication:	y = x * z	2.9	2.8	-3.4%
Division:	y = x / z	4.8	4.8	0.0%
Exponentiation:	y = x ** 2	2.7	2.8	3.7%
Modulus:	y = x mod z	13.5	13.3	-1.4%
Remainder:	y = x rem z	7.1	7.1	0.0%

The negative numbers in the table actually represent an *apparent improvement* in performance. However, they are within the measurement tolerance of 5%, and so are not statistically significant. Therefore the conclusion that might be drawn is that the size of the integer type has no effect on performance. Examination of the actual code, however, showed that the definitions of the integer subtypes "Int" and "Bigint" may be factors behind the results. These subtypes are defined (in package Global) as follows:

```
type Int_Type is new Integer;
subtype Int is Int_Type range -32_767.. +32767;
type Bigint_Type is new Integer;
subtype Bigint is Bigint_Type range -(2**30 -1 + 2**30) ..
(2**30 - 1 + 2**30);
```

Thus "Int" and "Bigint" are subtypes of a derived type that is derived from the same parent type (Integer). For the Verdix MC68030 compiler, this Integer type is a 32-bit quantity; the additional type Short_Integer is the analogous 16-bit quantity. The above definitions do not allow the compiler to choose type Short_-Integer for the 16-bit range of values. To allow the compiler to choose an appropriate representation, the definition of "Int", for example, could have been coded as follows:

```
type Int is range -32768 .. 32767;
```

This definition does not force the 32-bit Integer type to be used for 16-bit quantities.

References

- none

2.14 Private Types

Question: Is there a difference in performance between operations on objects of a private type and objects of a visible type?

Summary: The three suites of benchmark tests considered in this document provide no tests that explicitly answer this question. Use of private types within tests is incidental to the feature being tested.

Discussion: Private types are declared in the visible part of a package; their corresponding full type declarations occur in the specification after the word "private." There are certain operations that are only available to outside program units, while there are other operations that may only be performed within the package in which the private type is declared. Although none of the suites tests the performance of operations on objects of a private type, it is possible to construct tests to do so. In the reference cited below, tests were constructed to examine the difference between private and visible types and deferred versus non-deferred private types. For a Tartan Ada compiler for a MIL-STD-1750A target processor, there was no distinction made by the compiler between a private object and a visible object. There was a slight performance penalty incurred when deferred types were used because of the indirect access to these types.

References

[NASA]   NASA SEAS (Systems, Engineering, and Analysis Support) Program Technical Note. NASA Goddard Space Flight Center, Flight Dynamics Division / Code 552. April 10, 1990.

2.15 Records

Question: What is the performance of the various methods for assigning values to record objects?

Summary: Aggregate assignment of record component values is significantly more expensive than either record-to-record assignment or component-by-component assignment. There is also a performance penalty for using packed records. Depending on the record components, record-to-record assignment may or may not be faster than component-by-component assignment.

Discussion: A record is a composite object with named components, usually of different types. Values may be assigned to these components at run time (default assignment of values at compile-time is not considered here) in three different ways:

- a single record-to-record assignment statement
- a set of component-to-component assignment statements
- a single aggregate-to-record assignment

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: There are no AES or PIWG tests that address this issue; all results presented are from the ACEC suite. This first set of results shows that component-by-component assignment is about 27% more expensive than record-to-record assignment. Aggregate assignment is almost seven times more expensive than record assignment and five times more expensive than component-to-component assignment.

ACEC Test Results:

Configuration 1, ACEC Release 2.0, SSA Report "Record Assignment."

```
---------------------------------------------------------------------------

                         Language Feature Overhead

---------------------------------------------------------------------------
Record Assignment
---------------------------------------------------------------------------
Test        Execution    Bar                                      Similar
Name        Time         Chart                                    Groups
---------------------------------------------------------------------------
ss114         3.11       *****                                       |
ss115         3.95       ******                                      |
ss116        21.30       *********************************           |
---------------------------------------------------------------------------
                      Individual Test Descriptions
```

```
------------------------------------------------------------------------
    TYPE fields IS
        RECORD
            field_1 : string ( 1..3 ) ;
            field_2 : real ;
            field_3 : color ;
            field_4 : int range 1..10 ;
        END RECORD ;
      a, b : fields := ( "xxx", 0.0, red, 1 ) ;
------------------------------------------------------------
ss114 a := b ;                              -- Record assignment.
------------------------------------------------------------
ss115 a.field_1 := b.fie`d_1 ;
      a.field_3 := b.field_3 ;
      a.field_4 := b.field_4 ;
      a.field_2 := b.field_2 ;
-- Record component by component assignment (all fields).
------------------------------------------------------------
ss116 a := ( "xxx", one, hue, ei ) ;    -- Record assignment, aggregate.
------------------------------------------------------------------------
```

A comment in test 115 notes that an optimizing compiler could do a block transfer of contiguous fields, but that to recognize this would require the compiler to reorganize the component-by-component assignment statements.

Observation 2: This second group of ACEC results shows the effect of packing on both a set of component-by-component assignments and a record-to-record assignment. Mixed-mode assignments, where a conversion from packed to unpacked, or vice versa, is required, are also measured in this group of tests. In all cases packing has a significant effect on performance. By contrast with the previous group of results, this group shows record-to-record assignment to be *more* expensive than a set of component-by-component assignments.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, SSA Report "Field Assignments to Record (Packed, Unpacked)."

```
------------------------------------------------------------------------

                        Language Feature Overhead

------------------------------------------------------------------------
Field Assignments To Record (Packed, Unpacked)
------------------------------------------------------------------------
Test      Execution    Bar                              Similar
Name      Time         Chart                            Groups
------------------------------------------------------------------------
ss156        7.52      ********                           |
ss157        9.92      ***********                         |
ss160       10.60      ************                         |
ss161       11.50      *************                         |
ss158       22.40      ************************               |
ss159       31.10      ***********************************      |
------------------------------------------------------------------------
                        Individual Test Descriptions
------------------------------------------------------------------------
    TYPE descriptor IS      -- example from LRM 13.6
        RECORD    f1 :  small_int := 1 ;
                  f2 :  Boolean := True ;
                  f3 :  color := white ;
                  f4 :  med_int := 200 ;
```

```
                 f5  :  real := 1.0 ;
          END RECORD ;
        TYPE packed_descriptor IS NEW descriptor ;
        PRAGMA pack ( packed_descriptor ) ;
        a, b  :  descriptor ;
        c, d  :  packed_descriptor ;
-------------------------------------------------------------
ss156 a.f1 := integer ( ei ) ;      a.f2 := 11 /= mm ;
      a.f3 := hue ;                 a.f4 := 200 * integer ( ei ) ;
      a.f5 := yy ;                  -- field assignments to unpacked record
-------------------------------------------------------------
ss157 c.f1 := integer ( ei ) ;      c.f2 := 11 /= mm ;
      c.f3 := hue ;                 c.f4 := 200 * integer ( ei ) ;
      c.f5 := yy ;                  -- field assignment to packed record
-------------------------------------------------------------
ss158 a.f1 := integer ( ei ) ;      a.f2 := 11 /= mm ;
      a.f3 := hue ;                 a.f4 := 200 * integer ( ei ) ;
      a.f5 := yy ;
      c := packed_descriptor ( a ) ;-- unpacked to packed
-------------------------------------------------------------
ss159 c.f1 := integer ( ei ) ;      c.f2 := 11 /= mm ;
      c.f3 := hue ;                 c.f4 := 200 * integer ( ei ) ;
      c.f5 := yy ;
      b := descriptor ( c ) ;       -- unpack record
-------------------------------------------------------------
ss160 a.f1 := integer ( ei ) ;      a.f2 := 11 /= mm ;
      a.f3 := hue ;                 a.f4 := 200 * integer ( ei ) ;
      a.f5 := yy ;
      b := a ;                      -- unpacked record move
-------------------------------------------------------------
ss161 d.f1 := integer ( ei ) ;      d.f2 := 11 /= mm ;
      d.f3 := hue ;                 d.f4 := 200 * integer ( ei ) ;
      d.f5 := yy ;
      c := d ;                      -- packed record move
-------------------------------------------------------------------
```

References

- none

2.16 Rendezvous

Question: What are the performance characteristics of the various kinds of task rendezvous?

Summary: The larger the number of parameters passed in an entry call, the slower the rendezvous will be. The number of entries in a task has a significant effect on rendezvous performance whereas the number of tasks in a task set has virtually no effect. Simple rendezvous performance (no passed parameters) is slightly faster when the called task is waiting at an accept statement for an entry call. Equal-priority tasks yield rendezvous execution times that are about 25% faster than rendezvous times for tasks with different priorities because the order of execution of such tasks can be arranged to reduce the number of task switches. Use of a special pragma (**pragma** PASSIVE), available in the Verdix compiler, achieves a seven-fold reduction in rendezvous execution time.

Discussion: The rendezvous mechanism is the primary means of inter-task communication in the Ada programming language. It can take many forms, ranging from a simple synchronizing signal to an elaborate data-transfer operation with multiple choices for conditional or unconditional acceptance of the transferred data. Ideally, a benchmark, or set of benchmarks, to measure the performance of the Ada rendezvous would allow a user to generate a "performance envelope" for all the coding choices available. In practice, the benchmark suites discussed in this document provide specific data points within the performance envelope and give the user no control over the form of the actual tests. They do provide useful information, but in a format that often requires the user to examine the code of specific tests in order to understand the purpose of the tests and interpret the results. Because of the number of task rendezvous tests in the AES and ACEC suites, results from only a subset of these tests are presented below. The PIWG suite only contains eight tests to measure rendezvous performance so results from all of them are listed.

Configuration(s) tested:

1. **Host:** DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

   Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and 25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb zero-wait-state SRAM cache; 256-byte on-chip instruction cache; 256-byte on-chip data cache. The MVME141-1 board is paired with a Motorola MVME244-2 8Mb DRAM memory module board in a Motorola MVME945 VMEbus chassis.

   Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

Observation 1: AES Task Rendezvous Tests.

<u>AES Test Results:</u>

Configuration 1, DIY_AES Version 2.0, output from selected group K and L tests (TK02, TK03, TL02-TL8, and TL12-TL17).

Note: In the heading below, MASCOT (Modular Approach to System Construction, Operation, and Test) refers to a software design method than has been mandated by the UK Ministry of Defence since 1981.

K. Group K - MASCOT Tasking Tests

K.2. TK02

This test determines the cpu time taken to execute a simple rendezvous.

A benchmark was used to execute a simple rendezvous with a single scalar parameter. (A simple rendezvous is one where an accept statement accepts a normal entry call). The cpu time was measured when the caller was blocked first and when the accepting task was blocked first (this relies on pragma PRIORITY being effective).

        Caller blocked first   : 300us
        Accepting task blocked first : 287us

K.3. TK03

This test determines the cpu time taken to execute a simple rendezvous with guarded alternatives.

A benchmark was used to execute a simple rendezvous with a single scalar parameter. (A simple rendezvous is one where an accept statement accepts a normal entry call). The rendezvous is one where the called task has a selective wait with two guarded alternatives. The cpu time was measured when the caller was blocked first and when the accepting task was blocked first (this relies on pragma PRIORITY being effective).

        Caller blocked first   : 338us
        Accepting task blocked first : 344us

L. Group L - General Tasking Tests

L.2. TL02

This benchmark test determines the effect of idle tasks on performance. Tasks performing computations and idle tasks (i.e. tasks simply awaiting rendezvous) are used and the test executed several times, increasing the number of idle tasks each time. The percentage deterioration per idle task in system performance (as compared with no idle tasks being present) is examined.

```
        +-----------------+------------------+
        | Number of       | Deterioration    |
        | idle tasks      | per idle task    |
        +-----------------+------------------+
        |        1        |        0%        |
        |        5        |        0%        |
        |       10        |        0%        |
        |       20        |        0%        |
        +-----------------+------------------+
```

L.3. TL03

This benchmark test determines the effect of select
alternatives on performance.

This test measures the cpu time taken to select the first
alternative in a select statement containing two select
alternatives and also compares the cpu time taken in
selecting the first select alternative with the cpu time
taken in selecting the last select alternative in a
select statement containing 20 alternatives.

Number of Selections	Selection	Cpu Time
2	First	342us
20	First	418us
20	Last	441us

L.4. TL04

This benchmark test examines the effect on performance of
guards on entry statements.

The test measures the cpu time taken to select the first
alternative in a select statement containing two select
alternatives when simple boolean guards are present and
also compares the cpu time taken in selecting the first
select alternative with the time taken in selecting the
last select alternative in a select statement containing
20 alternatives when simple boolean guards are present.

Number of Selections	Selection	Cpu Time
2	First	345us
20	First	414us
20	Last	454us

L.5. TL05

This benchmark test examines the effect on performance of
passing parameters in rendezvous. The test compares the
cpu time taken for rendezvous with no parameters with
that for passing parameters. Various sized integer arrays
are used as parameters and average timings taken.

```
+-----------------+--------+---------+
|   Number of     |  Mode  |   Cpu   |
| Array Elements  |        |  Time   |
+-----------------+--------+---------+
|        0        |   -    |  174us  |
|       25        |   in   |  285us  |
|       25        | in out |  285us  |
|      100        |   in   |  285us  |
|      100        | in out |  285us  |
|     1000        |   in   |  285us  |
|     1000        | in out |  285us  |
|     2000        |   in   |  285us  |
|     2000        | in out |  285us  |
+-----------------+--------+---------+
```

L.6. TL06

This benchmark test determines the effect of multiple
entry clauses on performance.

This test compares the efficiency of using many small
tasks with single entry clauses against many large tasks
containing multiple entry clauses. 20 tasks/entry
statements are used. For the large task, efficiency of
selecting alternatives in reverse order is also compared
with that for making the selection in a straight order.

        20 tasks with single : 289us per rendezvous
         entry clauses

        Multiple entry clause :

         (in order)   426us per rendezvous
         (in reverse order)   425us per rendezvous

L.7. TL07

This benchmark test examines the effect of ordering on
entry clauses in a select statement.

This test compares the average cpu time taken to call a
select alternative when the alternatives are called in
order, with that to call a select alternative when the
alternatives are called in reverse order, for 10
alternatives.

        In order   : 382us per rendezvous
        In reverse order : 382us per rendezvous

L.8. TL08

This benchmark test determines the fairness of selective
wait statements.

A check is made that a task does not execute the else
alternative of a selective wait statement more than a
small number of times before a reschedule is forced.
Tasks of equal priority are used for the test.

The tasks were not given a fair distribution of time.

Note that the results of this test cannot be relied on if
pragma PRIORITY is not completely effective.

L.12. TL12

This test determines the rules for selecting open accept alternatives in the event that more than one can be chosen. If several rendezvous are possible and several accept alternatives can be selected, one is selected arbitrarily, [LRM 9.7.1 (6)].

The main task contained three open accept alternatives, three server tasks were also provided which looped indefinitely providing rendezvous opportunities with a given accept alternative. The test ascertained that, in 50 rendezvous, each accept alternative was used in sequence, the order being accept alternative 3, accept alternative 2 and accept alternative 1.

L.13. TL13

This test determines the rules for selecting open delay alternatives in the event that more than one can be chosen. An open delay alternative will be selected if no accept alternatives can be selected before the given delay has elapsed and that, if several delay alternatives can be selected, one is selected arbitrarily, [LRM 9.7.1 (8)].

The main task contained three open delay alternatives, each having a value of 20ms. No server tasks were provided and so there were no opportunities for rendezvous. The test ascertained that, in 50 attempted rendezvous, the same delay alternative was selected. This was delay alternative number 3.

L.15. TL15

This benchmark test examines the effect on performance of passing various numbers of parameters in rendezvous. The test compares the cpu time taken for rendezvous with no parameters with that for passing parameters. Various numbers of integer parameters are passed at rendezvous, the average timings being taken.

Number of Parameters	Cpu Time
0	174us
1	281us
10	289us
100	390us

L.16. TL16

This test determines the overheads of conditional entry calls which are not accepted, and selective waits which are not called.

For the conditional entry call test, there are two tasks. One makes repeated conditional entry calls, the other

```
contains an accept for the entry which is never executed.
The rendezvous is timed. The test ascertained that a
conditional entry call takes 41.5us.

For the selective wait test, there are two tasks. One
executes repeated selective waits, the other contains a
call to the selective wait entry which is never executed.
The rendezvous is timed. The test ascertained that a
selective wait takes 55.3us.
```

L.17. TL17

```
This test determines the overheads of using entry
families. Entry families define a "family" of entries,
each entry being distinguished by a different discrete
value.

Timings are performed for 5 ordinary entries, for an
entry family with 5 discrete values, for 10 ordinary
entries, and for an entry family with 10 discrete values.

The test used two tasks performing rendezvous with either
calls to ordinary entries, or to entry families.

The following results were obtained:
```

```
+---------------+----------+
|  Rendezvous  |   Cpu    |
|              |   Time   |
+---------------+----------+
|  5 ordinary  |  1.74ms  |
|  5 family    |  1.88ms  |
| 10 ordinary  |  3.76ms  |
| 10 family    |  4.00ms  |
+---------------+----------+
```

Observation 2: ACEC Task Rendezvous Tests.

<u>ACEC Test Results:</u>

Configuration 1, ACEC Release 2.0, raw output from selected tests:

- task3
- task4
- task23
- task24
- task26
- task30
- task31
- task41
- task42
- task43
- task47
- task_num_1
- task_num_5
- task_num_10
- task_num_15
- task_num_20
- task_num_25
- task_num_30
- task2_num_1
- task2_num_5
- task2_num_15
- task2_num_20
- task2_num_25
- task2_num30

The ACEC contains a great many tasking tests, and many of these tests measure rendezvous performance. To limit the scope of this observation, not all of the task rendezvous test results are reported here. All ACEC tasking results reported here come from the Ancillary Data section of the report generated by the Single System Analysis (SSA) tool. This section of the SSA report does not provide summary descriptions of tests and does not group test results into various categories (e.g., rendezvous with no passed parameters, rendezvous with N passed parameters). An attempt has been made to alleviate the problem for a subset of test results in this observation.

Simple Rendezvous: The results below are all from tests in which no parameters are passed in the rendezvous. Except for the last two tests listed, the calling and called tasks have different priorities as-

signed so that one or the other can be made to arrive at the rendezvous point first. The called task has the following structure:

```
TASK BODY resource IS
BEGIN
  LOOP
    ACCEPT request;
    ACCEPT release;
  END LOOP;
END resource;
```

The following is a summary description of the tests in this group:

TASK3:   caller arrives first, both tasks in same compilation unit

TASK23: callee arrives first, both tasks in same compilation unit

TASK24: caller arrives first, callee in subunit

TASK26: caller arrives first, callee in separate package

TASK41: callee arrives first, callee in separate package

TASK42: equal priority tasks, callee in separate package

TASK43: equal priority tasks, both in same compilation unit

Because the tasks in each of the last two tests above were of equal priority, it was possible for the compiler to execute them in an order that saved 1/4 of the task switches as compared with the execution order of the tasks in the preceding tests.

```
-----------------------------------------------------------------
task3 time per rendezvous =        251.8
-----------------------------------------------------------------
task23 time per rendezvous =       234.6
-----------------------------------------------------------------
task24 time per rendezvous =       252.0
-----------------------------------------------------------------
task26 time per rendezvous =       251.8
-----------------------------------------------------------------
task41 time per rendezvous =       235.6
-----------------------------------------------------------------
task42 time per rendezvous =       173.5
-----------------------------------------------------------------
task43 time per rendezvous =       172.0
-----------------------------------------------------------------
```

Selective wait with delay alternative: In the first test below, the called task has a select statement with a delay alternative that is not taken because the called task is ready to rendezvous with the calling task. The idea is that it should not be necessary for the delay to be set up and then cancelled. In the second test, the delay alternative is actually taken and then cancelled immediately when the calling task initiates the rendezvous.

```
------------------------------------------------------------------------
task30 time per rendezvous =      685.5
------------------------------------------------------------------------
task31 time per rendezvous =      761.6
------------------------------------------------------------------------
```

Bounded Buffer: In each of the following tests, elements are written to and read from a 10-element buffer. The buffer is in a task and reads and writes are performed by calling the appropriate entry. The elements are non-scalar in the first test (10-character strings) and scalar in the second test (single characters).

```
------------------------------------------------------------------------
task4 Time per rendezvous =       361.9
------------------------------------------------------------------------
task47 Time per rendezvous =      348.8
------------------------------------------------------------------------
```

The rendezvous times shown above were determined by dividing the elapsed time of a sequence of two reads and two writes by four. The same bounded-buffer tests were also featured in the main SSA report, where the elapsed time was reported. The SSA results are shown below.

```
------------------------------------------------------------------------

                       Runtime System Behavior

------------------------------------------------------------------------
Tasking-bounded Buffer With Scalar/nonscalar Parameter
------------------------------------------------------------------------
Test      Execution   Bar                                     Similar
Name      Time        Chart                                   Groups
------------------------------------------------------------------------
task47    1395.40     *********************************   |
task4     1447.40     *********************************   |
------------------------------------------------------------------------
                      Individual Test Descriptions
------------------------------------------------------------------------
On many systems, the time to process parameters in a rendezvous is a
small fraction of the time to perform the rendezvous proper.

SUBTYPE image IS String ( 1 .. 10 ) ;
x1 : image := "abcdefghij" ;
x2 : image := "0123456789" ;
y  : image ;
------------------------------------------------------------------------
task4 -- Bounded buffer with nonscalar parameter
   buffer.write ( x1 ) ;
   buffer.write ( x2 ) ;
   buffer.read ( y ) ;
   buffer.read ( y ) ;
------------------------------------------------------------------------
SUBTYPE image IS character RANGE 'A' .. 'Z';
x1 : image := 'B' ;
x2 : image := 'Y' ;
y  : image ;
------------------------------------------------------------------------
task47    -- Bounded buffer with scalar parameter
   buffer.write ( x1 ) ;
   buffer.write ( x2 ) ;
   buffer.read ( y ) ;
   buffer.read ( y ) ;
------------------------------------------------------------------------
```

Variable number of called tasks: Each test below (task_num_1, task_num_5, etc.) has a single calling task and many equal-priority called tasks; the actual number of called tasks is indicated in the test name. Each called task is of the form shown above for the simple rendezvous test group. The calling task makes a sequence of pairs of entry calls of the form

```
t1.request;
t1.release;

t2.request;
t2.release;
```

And so on, for each called task in the test. The idea is to see if rendezvous performance is affected by the number of tasks.

```
----------------------------------------------------------------------
task_num_1 time per rendezvous =          251.9
----------------------------------------------------------------------
task_num_5 time per rendezvous =          202.5
----------------------------------------------------------------------
task_num_10 time per rendezvous =         197.3
----------------------------------------------------------------------
task_num_15 time per rendezvous =         197.8
----------------------------------------------------------------------
task_num_20 time per rendezvous =         199.2
----------------------------------------------------------------------
task_num_25 time per rendezvous =         199.5
----------------------------------------------------------------------
task_num_30 time per rendezvous =         201.8
----------------------------------------------------------------------
```

Variable number of calling tasks: This test group is a variation on the group above. With only a single called task, the test is structured so that entry calls from many calling tasks are queued on the "request" entry of the called task before the "release" entry is called.

```
----------------------------------------------------------------------
task2_num_1 time per rendezvous =         234.1
----------------------------------------------------------------------
task2_num_5 time per rendezvous =         223.5
----------------------------------------------------------------------
task2_num_10 time per rendezvous =        228.5
----------------------------------------------------------------------
task2_num_15 time per rendezvous =        231.4
----------------------------------------------------------------------
task2_num_20 time per rendezvous =        235.6
----------------------------------------------------------------------
task2_num_25 time per rendezvous =        238.2
----------------------------------------------------------------------
task2_num_30 time per rendezvous =        243.2
----------------------------------------------------------------------
```

Observation 3: PIWG Task Rendezvous Tests.

<u>PIWG Test Results:</u>

Configuration 1, PIWG 12/12/87 Release, Tests T000001 through T000008.

```
Test Name:    T000001                    Class Name:   Tasking
CPU Time:       275.6  microseconds
Wall Time:      275.6  microseconds.     Iteration Count:   64
Test Description:
 Minimum rendezvous, entry call and return time
 1 task 1 entry , task inside procedure
 no select



Test Name:    T000002                    Class Name:   Tasking
CPU Time:       275.6  microseconds
Wall Time:      275.6  microseconds.     Iteration Count:   64
Test Description:
 Task entry call and return time measured
 One task active, one entry in task, task in a package
 no select statement



Test Name:    T000003                    Class Name:   Tasking
CPU Time:       278.3  microseconds
Wall Time:      278.3  microseconds.     Iteration Count:   32
Test Description:
 Task entry call and return time measured
 Two tasks active, one entry per task, tasks in a package
 no select statement



Test Name:    T000004                    Class Name:   Tasking
CPU Time:       330.6  microseconds
Wall Time:      330.6  microseconds.     Iteration Count:   16
Test Description:
 Task entry call and return time measured
 One tasks active, two entries, tasks in a package
 using select statement



Test Name:    T000005                    Class Name:   Tasking
CPU Time:       279.0  microsecords
Wall Time:      278.8  microseconds.     Iteration Count:   4
Test Description:
 Task entry call and return time measured
 Ten tasks active, one entry per task, tasks in a package
 no select statement



Test Name:    T000006                    Class Name:   Tasking
CPU Time:       401.0  microseconds
Wall Time:      400.8  microseconds.     Iteration Count:   4
Test Description:
 Task entry call and return time measurement
 One task with ten entries , task in a pack/ e
```

```
               one select statement, compare to T000005



        Test Name:    T000007                   Class Name:   Tasking
        CPU Time:      171.9  microseconds
        Wall Time:     171.7  microseconds.      Iteration Count:    64
        Test Description:
         Minimum rendezvous, entry call and return time
         1 task 1 entry
         no select



        Test Name:    T000008                   Class Name:   Tasking
        CPU Time:      609.4  microseconds
        Wall Time:     609.4  microseconds.      Iteration Count:    32
        Test Description:
         Measure the average time to pass an integer
         from a producer task through a buffer task
         to a consumer task
```

The Verdix compiler provides a "PASSIVE" pragma that enables certain kinds of tasks to be optimized for runtime performance. For comparison, the results of modified PIWG T tests are shown below. The T tests were modified by inserting a "pragma PASSIVE" in the specification of each task. The VADS documentation states that this pragma is not allowed in all cases; for example, tests T000001 and T000007 generated a compile-time warning of the form:

```
        20:   task T1 is
        21:     entry E1 ;
        22:     pragma PASSIVE;
     A -------^
     A:warning: Appendix F: PASSIVE only allowed for a task declared in a
                 library package
        23:   end T1 ;
```

For this reason, the pragma was only put in the specification of the BUFFER_TYPE task type in test T000008.

```
        Test Name:    T000001                   Class Name:   Tasking
        CPU Time:      275.6  microseconds
        Wall Time:     275.8  microseconds.      Iteration Count:    64
        Test Description:
         Minimum rendezvous, entry call and return time
         1 task 1 entry , task inside procedure
         no select



        Test Name:    T000002                   Class Name:   Tasking
        CPU Tin :      36.3   microseconds
        Wall Time:     36.3   microseconds.      Iteration Count:    256
        Test Description:
         Task entry call and return time measured
         One task active, one entry in task, task in a package
         no select statement
```

```
Test Name:    T000003                    Class Name:  Tasking
CPU Time:        37.9  microseconds
Wall Time:       37.9  microseconds.      Iteration Count:   128
Test Description:
 Task entry call and return time measured
 Two tasks active, one entry per task, tasks in a package
 no select statement




Test Name:    T000004                    Class Name:  Tasking
CPU Time:        40.3  microseconds
Wall Time:       40.2  microseconds.      Iteration Count:   128
Test Description:
 Task entry call and return time measured
 One tasks active, two entries, tasks in a package
 using select statement




Test Name:    T000005                    Class Name:  Tasking
CPU Time:        35.5  microseconds
Wall Time:       35.5  microseconds.      Iteration Count:   32
Test Description:
 Task entry call and return time measured
 Ten tasks active, one entry per task, tasks in a package
 no select statement




Test Name:    T000006                    Class Name:  Tasking
CPU Time:        53.4  microseconds
Wall Time:       53.4  microseconds.      Iteration Count:   32
Test Description:
 Task entry call and return time measurement
 One task with ten entries , task in a package
 one select statement, compare to T000005




Test Name:    T000007                    Class Name:  Tasking
CPU Time:       171.9  microseconds
Wall Time:      172.0  microseconds.      Iteration Count:   64
Test Description:
 Minimum rendezvous, entry call and return time
 task 1 entry
 no select




Test Name:    T000008                    Class Name:  Tasking
CPU Time:       151.7  microseconds
Wall Time:      151.7  microseconds.      Iteration Count:   128
Test Description:
 Measure the average time to pass an integer
 from a producer task through a buffer task
 to a consumer task
```

References

- none

Appendix A   BSY-2 Performance Questions

This appendix describes how the performance questions addressed in this report were selected. There are two primary sources for questions (in order of examination):

1.   The BSY-2 SSP Style Guide

2.   SEI experience

The Style Guide was examined first and a list of questions was generated from it. A supplementary list of questions was then prepared to cover additional topics based on the experiences of the authors and reviewers within the SEI.

The question lists were developed to be inclusive, particularly in analyzing the Style Guide. Only trivial questions were edited from these preliminary lists.

Each question was designed to quantify the performance of alternate programming constructs which a programmer or system designer might reasonably select to implement a program or system. In general these are choices between alternate Ada constructs.

The lists were then merged and the most significant items were selected for further examination. In preparing the merged list the REST Project staff considered two primary criteria:

- The potential performance payoff between alternatives.
- The availability of performance tests and data to answer the questions.

The question list was then reviewed within the SEI for relevance and completeness.

A.1   Questions from the SSP Ada Style Guide

The Software Standards and Procedures Manual (SSP) for the AN/BSY-2 contains an Ada Style Guide which will control the format for coding Ada software. The style guide emphasizes readability, consistency and maintainability. It therefore specifies how variables and Ada statements are to be constructed, module size, format of statements and specifies usage rules for a number of language features. The Ada Style Guide does not specifically discuss execution efficiency for these constructs.

The SSP Ada Style Guide was reviewed, and any recommendations which represented choices containing possible performance trade offs were isolated and the relevant performance questions were formulated. These questions are the primary basis for this report. The list included here is the complete list derived from the Style Guide. It was subsequently edited and the questions collated into the final list.

A.1.1   Performance Issues Relating to the Ada Style Guide
The Ada Style Guide is relatively brief and does not specifically address Ada performance issues. However, some performance questions are relevant to specific rules laid down by the Style Guide:

1.   Sections 10.3.4 and 10.3.5.7 specify the use of **digits** N rather than using the predefined types SHORT_FLOAT, FLOAT, and LONG_FLOAT.

   1.1.   Does performance differ if predefined types are used?

1.2. How does the compiler select the underlying representations when numeric representations are set by the user?

1.3. What is the relationship between requested accuracy and performance? Is the relationship a smooth curve or discontinuous?

2. Section 10.3.5.1 specifies the use of enumeration types for clarity rather than code values or strings.

2.1. What are the performance characteristics of enumerated types versus data representations using strings, characters and integers?

3. Section 10.3.5.5.b specifies that the attributes 'RANGE or 'FIRST and 'LAST are to be used in preference to constant values in setting bounds and ranges for loops and similar constructs.

3.1. What are the performance trade offs between ranges provided by attributes and by constant values?

4. Section 10.3.5.11 forbids the use of anonymous types to define arrays.

4.1. Is the performance of arrays using anonymous types different from typed arrays?

5. Section 10.4.3.2.b specifies that array aggregates should be used in place of explicit loops "wherever applicable."

5.1. What are the performance characteristics of array aggregates and corresponding loops?

6. Section 10.5.4.a states that a case statement should be used "when a selection is based on the value of a single variable or expression of a discrete type other than Boolean."

6.1. What are the performance trade offs between the case statement and other logical selectors?

6.2. Does the performance of logical selectors vary with number of selection alternatives?

7. Section 10.7.3.a specifies that a subroutine should normally contain 100 or fewer executable lines, and never more than 200.

7.1. Does the extra call overhead meaningfully increase the execution time of programs divided into smaller modules compared to a monolithic design?

7.2. Is **pragma** INLINE effective?

8. Section 10.6.3.e specifies that **pragma** INLINE should only be used for procedures and functions that are internal to the body of a package, task, procedure or function.

8.1. How does the use of **pragma** INLINE affect runtime performance?

9. Section 10.9.2.a specifies that task types should be used instead of multiple task definitions performing the same function.

9.1. What are the comparative performance values for task types versus multiple task definitions?

10. Section 10.9.6.b specifies that all programs using time should use the CALENDAR.TIME type or DURATION type except when "more precision" is needed.

   10.1. What is the accuracy of types CALENDAR.TIME and DURATION?

   10.2. What accuracy can be expected using alternate time formats?

   10.3. What is the performance of alternative time formats?

11. Section 10.9.8.e allows tasks to be declared without any priority. Section 10.9.8 does not state if tasks with and without priority may be mixed.

   11.1. What is the default priority of a task?

   11.2. Does the behavior of a task with a default priority vary at all from an equivalent task with an explicit priority?

   11.3. What happens when tasks with and without priority are mixed in a single program?

12. Section 10.9.11 forbids the use of **pragma** SHARED.

   12.1. What is the efficiency of shared variables specified by **pragma** SHARED?

13. Section 10.11.7.a specifies that runtime checks will not be suppressed unless suppression is required to achieve acceptable program efficiency.

   13.1. How is performance changed by suppressing runtime checks?

   13.2. What optimizations does the compiler perform for runtime checking?

   13.3. How is performance affected by suppressing individual runtime checks?

   13.4. Does simulating selected runtime checks by explicit comparisons offer any performance advantage?

14. Section 10.12 describes the use of generic units.

   14.1. What is the comparative performance of generic and nongeneric units?

15. Section 10.12.2.a specifies that a generic subroutine should normally contain 100 or fewer executable lines, and never more than 200.

   15.1. Does the extra call overhead meaningfully increase the execution time of programs divided into smaller generic modules compared to a monolithic design?

   15.2. Is **pragma** INLINE effective for generic routines?

A.2 SEI Additional Questions

The SEI supplemented the list of questions from the SSP Style Guide with some additional questions. These are generally intended to augment the recommendations from the style guide or to explore issues that the style guide does not cover.

A.2.1 SEI Questions

1. Compiler Optimizations

    1.1. What are the effects of different levels of optimization?

    1.2. What specific optimizations are performed? (Relate to optimization levels if possible.)

    1.3. Does optimization minimize or eliminate unnecessary runtime checking?

2. Device Interfacing

    2.1. Can an access type be used to map a data structure to a real device address?

    2.2. Can a record representation clause be used to specify the structure of device registers?

    2.3. Can unchecked type conversion (via generic function UNCHECKED_CONVERSION) be used to generate real device addresses from integer representations of such addresses?

    2.4. Is package LOW_LEVEL_IO implemented? Is it a viable alternative to using address clauses, record representation clauses, and unchecked type conversion?

    2.5. Is interfacing to other languages (in particular, assembler language) supported? Is the performance different from the all-Ada approach?

    2.6. How are interrupts handled?

       2.6.1. Can an address clause be used to map an interrupt entry to a real interrupt vector?

       2.6.2. What is the interrupt latency for handlers written in Ada?

       2.6.3. Can interrupt handlers written as Ada tasks be optimized to provide performance comparable with that of other types of handlers?

       2.6.4. At what priority level does the interrupt entry execute?

       2.6.5. At what priority level does the task body outside the interrupt entry execute?

3. Exception Handling

    3.1. Does the presence of an exception handler affect runtime performance?

    3.2. What is the runtime cost of raising and propagating an exception?

4. Loop Control

    4.1. Do different loop constructs vary in efficiency?

5. Numeric Operations.

    5.1. What is the performance impact of using double-precision versus single precision arithmetic?

5.2. Is there a performance difference between predefined numeric types and user-defined numeric types?

5.3. Is the math library efficient?

6. Pragmas

6.1. What pragmas are supported?

6.2. How does **pragma** ELABORATE affect the performance of a program?

6.3. Does inlining of subprograms via **pragma** INLINE improve performance? By how much?

7. Program Structure

7.1. What are the performance characteristics of subprograms?

7.1.1. What is the overhead of calling subprograms with various numbers/-modes/types of parameters?

7.2. What are the performance characteristics of generic objects? How do they compare with the equivalent non-generic objects?

7.3. How does the locality of data and procedures affect performance?

7.3.1. What is the performance effect of declaring data locally, within the package, in other packages, in library units?

7.3.2. What is the overhead of calling subprograms that are in the same unit, in different units, in different packages?

7.3.3. What effect does the use of private and limited private types have on performance?

7.3.4. What is the overhead of calling subprograms that are in subunits (i.e., separately compiled)?

8. Representation of Data

8.1. What are the performance and memory size consequences of using representation attributes to vary numeric characteristics?

8.2. What are the performance and memory size consequences of using representation attributes for selected data types (e.g., arrays, records, strings)?

8.3. Are there performance and memory size differences between private and public data types?

9. Runtime Checking

9.1. Does runtime checking impose a significant performance overhead?

9.2. What is the performance of UNCHECKED_CONVERSION versus explicit type conversion?

10. Tasking

10.1. What are the performance characteristics of a tasking program?

10.1.1. Does the presence or absence of an explicitly-defined priority affect a task's performance?

10.1.2. How long does it take to create/terminate a task?

10.1.3. Is there a difference in the performance of dynamically-created and statically-created tasks?

10.1.4. Does the presence of an exception handler in a task affect that task's performance?

10.2. What are the performance characteristics of task rendezvous?

10.2.1. How is rendezvous performance affected by the number of tasks?

10.2.2. How is rendezvous performance affected by the number of entries in the task(s)?

10.2.3. How is rendezvous performance affected by the presence of guards in the rendezvous?

10.3. What are the characteristics of task set scheduling?

10.3.1. What algorithm(s) is(are) used to schedule a task set?

10.3.1.1. What algorithm is used to schedule a task set with explicitly defined priorities?

10.3.1.2. What algorithm is used to schedule a task set without explicitly defined priorities?

10.3.1.3. What algorithm is used to schedule a task set with a mixture of explicitly defined priorities and undefined priorities?

10.3.2. Is the runtime system preemptive and priority-based?

10.3.3. Is blocking minimized?

10.3.4. Is priority inversion avoided?

10.3.5. Is I/O interleaved with task execution or is it a blocking effect?

11. Time Management

11.1. What is the resolution of CALENDAR.CLOCK?

11.2. What is the resolution of the **delay** statement?

11.3. What is the resolution of the type DURATION (DURATION'SMALL)?

11.4. What is the overhead of reading CALENDAR.CLOCK?

11.5. What is the overhead of performing calculations with types TIME and DURATION?

11.6. Is CALENDAR.CLOCK subject to drift or jitter? If so, how much?

A.3 Combined Questions List

Performance questions from all sources were combined into a single list, organized by subject. Individual questions from this list are then addressed in the body of the report.

Note: SSP indicates a question from the BSY-2 SSP Style Guide; SEI indicates a question provided by the authors.

A.3.1 Merged List of Questions

1. Tasking

   1.1. Task Priority

      1.1.1. What is the default priority of a task? [SSP 11.1]

         1.1.1.1. A master task (the task created for the main program)?

         1.1.1.2. A library task (a task created in a library unit)?

         1.1.1.3. "Vanilla" tasks (tasks created within a program either by declaration or dynamically)?

   1.2. Task Set Scheduling

      1.2.1. Algorithm(s)

         1.2.1.1. What algorithm is used to schedule a task set with explicitly defined priorities? [SSP 11.2 and SEI 10.3.1.1]

         1.2.1.2. What algorithm is used to schedule a task set without explicitly defined priorities? [SSP 11.2 and SEI 10.3.1.2]

         1.2.1.3. What algorithm is used to schedule a task set with a mixture of explicitly defined priorities and undefined priorities? [SSP 11.3 and SEI 10.3.1.3]

      1.2.2. Is the runtime system preemptive and priority-based? [SEI 10.3.2]

      1.2.3. Is blocking minimized? [SEI 10.3.3]

      1.2.4. Is priority inversion avoided? [SEI 10.3.4]

      1.2.5. Is I/O interleaved with task execution or is it a blocking effect? [SEI 10.3.5]

   1.3. Task Performance

      1.3.1. What are the comparative performance values for task types versus multiple task definitions? [SSP 9.1 and SEI 10.1.3]

      1.3.2. Does the presence or absence of an explicitly-defined priority affect a task's performance? [SEI 10.1.1]

      1.3.3. How long does it take to create/terminate a task? [SEI 10.1.2]

1.3.4. Does the presence of an exception handler in a task affect that task's performance? [SEI 10.1.4]

1.4. Rendezvous

1.4.1. How is rendezvous performance affected by the number of tasks? [SEI 10.2.1]

1.4.2. How is rendezvous performance affected by the number of entries in the task(s)? [SEI 10.2.2]

1.4.3. How is rendezvous performance affected by the presence of guards in the rendezvous? [SEI 10.2.3]

2. Compiler Optimizations

2.1. What are the effects of different levels of optimization? [SEI 1.1]

2.2. What specific optimizations are performed? (Relate to optimization levels if possible.) [SEI 1.2]

2.3. Does optimization minimize or eliminate unnecessary runtime checking? [SEI 1.3]

3. Loop Control

3.1. Do different loop constructs vary in efficiency? [SEI 4.1]

3.2. What are the performance trade offs between loop limits set by constants and by attributes (e.g., 'FIRST, 'LAST, or 'RANGE)? [SSP 3.1]

4. Logical Testing

4.1. What are the performance trade offs between the case statement and other logical tests? [SSP 6.1]

4.2. Does the performance of logical selectors vary with the number of selection alternatives? [SSP 6.2]

5. Data Representation

5.1. Predefined Types

5.1.1. Is there a difference in performance between predefined and user defined types? [SSP 1.1]

5.1.2. Time

5.1.2.1. What is the accuracy of types CALENDAR.TIME and DURATION? [SSP 10.1 and SEI 11.3]

5.1.2.2. What accuracy can be expected using alternate time formats? [SSP 10.2]

5.1.2.3. What is the performance of alternative time formats? [SSP 10.3]

5.2. User Defined Types

5.2.1. Numeric Types

5.2.1.1.   Is there a performance difference between predefined numeric types and user-defined numeric types? [SEI 5.2]

5.2.1.2.   How are underlying types selected for number representations? [SSP 1.2]

5.2.1.3.   What is the relationship between requested accuracy and performance? Is the relationship a smooth one or discontinuous? [SSP 1.3]

5.2.1.4.   What is the performance impact of using double-precision versus single precision arithmetic? [SEI 5.1]

5.2.1.5.   Is the math library efficient? [SEI 5.3]

5.2.2. Enumeration Types

5.2.2.1.   How does the performance of enumeration types compare with that of equivalent representations using strings or code values? [SSP 2.1]

5.2.3. Private Types

5.2.3.1.   Are there performance and memory size differences between private and public data types? [SEI 8.3]

5.3. Arrays

5.3.1. Definitions

5.3.1.1.   Is the comparative performance of arrays different when they are defined anonymous types and named types? [SSP 4.1]

5.3.2. Assignment

5.3.2.1.   What are the performance characteristics of array aggregates and corresponding loops? [SSP 5.1]

5.4. Representation of Data

5.4.1.   What are the performance and memory size consequences of using representation attributes to vary numeric characteristics? [SEI 8.1]

5.4.2.   What are the performance and memory size consequences of using representation attributes for selected data types (e.g., arrays, records, strings)? [SEI 8.2]

5.5. Type Conversion

5.5.1.   What is the performance of UNCHECKED_CONVERSION versus explicit type conversion? [SEI 9.2]

6.   Generics

6.1. What is the comparative performance of generic and nongeneric units? [SSP 14.1]

6.2. Does the extra call overhead meaningfully increase the execution time of programs divided into smaller generic modules compared to a monolithic design? [SSP 15.1]

7.   Program Structure

7.1. Subprograms

7.1.1. Does the call overhead meaningfully increase the execution time of programs divided into smaller modules compared to a monolithic design? [SSP 7.1]

7.1.2. What is the overhead of calling subprograms with various numbers/modes/types of parameters? [SEI 7.1.1]

7.2. What are the performance characteristics of generic objects? How do they compare with the equivalent non-generic objects? [SEI 7.2]

7.3. Locality of data and procedures

7.3.1. What is the performance effect of declaring data locally, within the package, in other packages, in library units? [SEI 7.3.1]

7.3.2. What is the overhead of calling subprograms that are in the same unit, in different units, in different packages? [SEI 7.3.2]

7.3.3. What effect does the use of private and limited private types have on performance? [SEI 7.3.3]

7.3.4. What is the overhead of calling subprograms that are in subunits (i.e., separately compiled)? [SEI 7.3.4]

8.   Pragmas

8.1. What pragmas are supported? [SEI 6.1]

8.2. **pragma** ELABORATE

8.2.1. How does **pragma** ELABORATE affect the performance of a program? [SEI 6.2]

8.3. **pragma** INLINE

8.3.1. Is **pragma** INLINE effective? [SSP 7.2 and SEI 6.3]

8.3.2. What is the runtime performance effect of **pragma** INLINE? [SSP 8.1 and SEI 6.3]

8.3.3. Is **pragma** INLINE effective for generic routines? [SSP 15.2]

8.4. pragma SHARED

8.4.1. What is the efficiency of shared variables specified by pragma SHARED? [SSP 12.1]

8.5. pragma SUPPRESS

8.5.1. How is performance changed by suppressing all constraint checks? [SSP 13.1 and SEI 9.1]

8.5.2. How is performance changed by suppressing individual constraint checks? [SSP 13.3 and SEI 9.1]

8.5.3. Does simulating constraint checks by explicit comparison offer any performance advantage? [SSP 13.4 and SEI 9.1]

8.5.4. What optimizations does the compiler perform for constraint checking? [SSP 13.2]

9. Device Interfacing

9.1. Can an access type be used to map a data structure to a real device address? [SEI 2.1]

9.2. Can a record representation clause be used to specify the structure of device registers? [SEI 2.2]

9.3. Can unchecked type conversion (via generic function UNCHECKED_CONVERSION) be used to generate real device addresses from integral representations of such addresses? [SEI 2.3]

9.4. Is package LOW_LEVEL_IO implemented? Is it a viable alternative to using address clauses, record representation clauses, and unchecked type conversion? [SEI 2.4]

9.5. Is interfacing to other languages (in particular, assembler language) supported? Is the performance different from the all-Ada approach? [SEI 2.5]

9.6. Interrupts

9.6.1. Can an address clause be used to map an interrupt entry to a real interrupt vector? [SEI 2.6.1]

9.6.2. What is the interrupt latency for handlers written in Ada? [SEI 2.6.2]

9.6.3. Can interrupt handlers written as Ada tasks be optimized to provide performance comparable with that of other types of handlers? [SEI 2.6.3]

9.6.4. At what priority level does the interrupt entry execute? [SEI 2.6.4]

9.6.5. At what priority level does the task body outside the interrupt entry execute? [SEI 2.6.5]

10. Exception Handling

    10.1. Does the presence of an exception handler affect runtime performance? [SEI 3.1]

    10.2. What is the runtime cost of raising and propagating an exception? [SEI 3.2]

11. Time Management

    11.1. What is the resolution of CALENDAR.CLOCK? [SEI 11.']

    11.2. What is the resolution of the **delay** statement? [SEI 11.2]

    11.3. What is the resolution of the type DURATION (DURATION'SMALL)? [SEI 11.3]

    11.4. What is the overhead of reading CALENDAR.CLOCK? [SEI 11.4]

    11.5. What is the overhead of performing calculations with types TIME and DURATION? [SEI 11.5]

    11.6. Is CALENDAR.CLOCK subject to drift or jitter? If so, how much? [SEI 11.6]

A.4 Editing the Questions List

The merged list of questions was edited to include questions which could be quantitatively analyzed and which had a high explanatory value. The questions were then rewritten for additional clarity and to exploit available data.

A.4.1 Final Questions List

This list is organized alphabetically by topic.

1. **Arrays:** What are the performance characteristics of array aggregate assignments and corresponding loop constructs?

2. **Check Suppression:** How does performance change when checks are turned off?

3. **Data Location:** What is the performance effect of declaring data locally or outside the executing scope?

4. **Enumeration Types:** How does the performance of operations on objects of an enumeration type compare with the performance of an equivalent representation using strings or numeric values?

5. **Exceptions:** What are the performance consequences of providing exception handling capabilities?

6. **Generic Units:** What is the comparative performance of generic and non-generic units?

7. **Inlining of Procedures:** What is the effect of inlining procedures and generic procedures?

8. **Logical Tests:** What are the performance trade offs between the **case** statement and **If** statement?

9. **Loop Efficiency:** Do different loop constructs vary in efficiency?

10. **Module Size:** Is the performance of a program divided into modules different from a monolithic design?

11. **Optimization Options:** What are the effects of different optimization levels?

12. **Precision:** What are the performance differences between single-precision and extended-precision numeric operations?

13. **Private Types:** Is there a difference in performance between operations on objects of a private type and objects of a visible type?

14. **Records:** What is the performance of the various methods for assigning values to record objects

15. **Rendezvous:** What are the performance characteristics of the various kinds of task rendezvous?

Appendix B Benchmark Sources

Information on how to obtain the source code and documentation of the benchmark suites used in this report is presented below.

B.1 The Ada Evaluation System

The Ada Evaluation System (AES) may be obtained from the British Standards Institution at the following address:

> Software Product Services
> Software Engineering Department
> BSIQA
> P.O. Box 375
> Milton Keynes MK14 6LL
> United Kingdom
> Tel. 0908 220908
> UUCP: sed@bsiqa.uucp
> (Internet: bsiqa!sed@uunet.uu.net)

As of this writing, the current version is the DIY-MAPSE-01 version. It is available at a cost of 3000 pounds sterling. BSI also offers a validation service at a cost of 24,000 pounds sterling.

The Ada Evaluation System (AES) will be merged with the Ada Compiler Evaluation Capability (ACEC) under a joint agreement between the Ministry of Defence of the United Kingdom and the Department of Defense of the United States that was signed in June of 1991. The merged product will be released as version 4.0 of the ACEC; as of this writing, the expected release time is the third or fourth quarter of 1993.

B.2 The Ada Compiler Evaluation Capability

The Ada Compiler Evaluation Capability (ACEC) may be obtained from

> Data and Analysis Center for Software (DACS)
> P.O. Box 120
> Utica, NY 13503
> Tel. (315) 734-3696
> Internet: dacs-info@kaman.com

As of this writing, the current release of the ACEC is 3.0. There are three documents: the User's Guide, the Reader's Guide, and the Version Description Document. The total cost for the software and documentation is 100 US dollars. (Release 3.0 of the ACEC is not the merged AES-ACEC product referred to above.)

B.3 The PIWG Benchmarks

The ACM Performance Issues Working Group (PIWG) benchmarks may be obtained in one of three ways:

- Via anonymous ftp (Internet file transfer protocol) from the **ajpo.sei.cmu.edu** machine. Users should issue the command "ftp ajpo.sei.cmu.edu" and log in using the word "anonymous" as the login name and an identifying string (e.g., the user's e-mail address) as password. Change directory ("cd" command) to the "public/piwg/piwg_11_92" directory and use the ftp file-transfer commands to retrieve the files.

- Via the PIWG bulletin board. Ideally, users should access this from a PC (rather than a dumb terminal) using a modem capable of sending and receiving at 1200 baud or higher. The number of the bulletin board is (412) 268-7020. Once connected to the bulletin board, users will be able to navigate their way around the system using simple menus that the system provides. The point of contact for this service is Gene Rindels, (412) 268-6728.

- Via a written request or telephone request to the following service:

  PIWG Distribution
  Software Engineering Institute
  Carnegie Mellon University
  Pittsburgh, PA 15213-3890
  Tel. (412) 268-7787

As of this writing, the current release of the PIWG suite is the one known as the 11/92 release. There is no charge for the PIWG benchmarks. Documentation for the PIWG benchmarks consists principally of the READ.ME file distributed with the suite and comments in the individual test programs and command files. There is also additional information about the PIWG suite in the Winter 1990 special edition of Ada Letters (Vol. X, No. 3, special edition on Ada Performance Issues).

Appendix C   Question Format and Instructions

Each of the questions covered by this report is prepared using a standard format. This appendix presents that format.

C.1   Blank Question Entry

The blank question entry shows the headings used for a performance entry.

3.1

Question: ?

Summary:

Discussion:

Configuration(s) tested:

    1.

Observation 1:

Observation 2:

References

    •

C.2   Instructions for Filling in Questions

Each entry in the performance and style report addresses a topic where we believe Ada programmers or system designers will have to make choices which affect performance. The entries offer guidance on how to program for high performance. Each entry should be designed to be read independently of all other entries.[1] In addition to presenting the conclusions of the author(s), sufficient raw data should be included to support all conclusions drawn and references provided so that the reader can repeat the experiments, both with the tested configuration and using hardware and software variants and upgraded versions.

C.2.1   Topic

The topic serves as a title and alphabetizing key for the entry. The topic rephrases the question to put the keywords in order of importance. For example, the question "Do different loop constructs vary in efficiency?" is translated to the topic "Loop Efficiency". If ordering the keywords in order of importance

---

1. However, entries should freely reference outside sources and appendices included in this report which can be shared between entries, especially where it would be tedious to repeat the information for each entry. Related entries should be mentioned, as a cross reference, but their contents should not be included.

---

conflicts with clearly conveying the sense of the topic, the topic should be phrased for clarity, as it is assumed that cross referencing in the subject index will make up for any "unnatural" placements. The topic should be composed to allow readers to browse through the report and identify the interesting sections.

Question: ?

The question poses the topic for the entry. It is normally phrased to indicate the area of interest and the coding choices which the entry will address. (Topics considered for this report will normally discuss performance issues where the application developer has a choice or choices about the programming idiom. The report tells the reader what the performance consequences of each choice is.)

An example question is "Do different loop constructs vary in efficiency?" where the programming idiom is loop constructs (which are too numerous to enumerate in the question) and the phrase "vary in efficiency" is tagged on to serve as a reminder to the reader that they will receive information on run time efficiency. In answering the question, the author must list loop constructs, indicate which were tested, and then draw conclusions about "efficiency".

Questions must be both short and readily comprehensible. While the question should accurately reflect the contents of the entry, it should lean towards generality rather than being highly specialized. While this will lead to partial answers, this is neither unexpected nor undesirable. Limits of available test data from standard benchmark suites should be called out, and, if required, supplemented by customized tests (or via improvement to the standard suites).

Summary:

The summary is a **brief** statement of the question's answer. It should state the answer by:

- Declaring whether the question **could** be answered—there may be cases where a valid and interesting topic could not be analyzed for lack of data.
- Defining the "winners" and/or the "losers" for the question, that is, which alternative is the most efficient.

  However, there may be several ways of selecting winners and losers (for example, the fastest choice may not be the most space efficient). These qualifications must be mentioned.

  In some cases the difference may not be significant or consistent, which should then be called out in the answer. Lengthy explanations should be reserved for later sections of the entry.

- Any significant limitations should be mentioned. For instance, if the available data misses several alternatives, a qualification should be made. However general caveats such as "this conclusion is based on limited data" should not be made in the summary.

Ideally, the summary should be a single sentence. In any case it should be limited to a short paragraph. If additional qualification is required, it should be a forward reference to the discussion or observation sections.

Discussion:

The discussion section examines issues about the question and the answer for the whole of the entry. Issues which are appropriately included in the discussion section should:

- span the individual observations.
- modify or amplify the conclusions which are made in the summary section.
- suggest additional data which were either not available or which could not be obtained.

While results from individual observations can be mentioned, this should only be done to highlight especially interesting or significant results. For instance, if it is generally true that the **case** statement is faster than nested **if** statements, but in one special case the **if** statement is superior, this could appropriately be mentioned, but the details should continue to be included in the observation.

The discussion section should also be used to resolve inconsistencies between observations, but not to enumerate errors which are confined to a single observation. Thus, a discussion section would appropriately describe why the timings obtained from the vendor's literature in Observation #2 do not match those obtained from benchmark testing in Observation #1. However, if the various tests within Observation #1 disagree, this should be analyzed within the observation itself. The intra-observational inconsistency would be mentioned in the discussion only when it was not resolved for the observation and consequently affected conclusions (as mentioned in the preceding paragraph).

An example of when to include intra-observational inconsistencies: if, in testing loop execution speed, an AES test concludes that a **while** loop is faster than a **for** loop and an ACEC test rates the **for** loop faster, this issue is appropriately mentioned in the discussion section. However, if investigation shows that the AES test allowed the compiler to optimize the **while** loop to be null, the inconsistency has been resolved within the observation and should not be mentioned in the discussion section.

Configuration(s) tested:

1.   *Each configuration is assigned a number, allowing it to be conveniently referenced (in FrameMaker, use paragraphs from the **enumerate** family).*

The configuration(s) tested section lists the tested host and target configuration. A configuration may have been used for one or several observations. The configuration should contain both generic descriptions (e.g., "a Motorola 68020") and specific items (e.g., "a Motorola Microsystems MVME 133A-20 single board computer") which completely characterize the hardware and software used in the test: The configuration does not describe the test(s) performed on the configuration. The tests are described in the observations sections.

In cases where reference material contains test results from real systems, the tested configuration should be described in this section.

A configuration ideally contains the information listed below (it is recognized that some configurations may be incomplete in some areas). Since there are numerous important characteristics of microprocessor systems, this list may be augmented by additional information where appropriate (for example, a tested multiprocessor system might need to specify tested topology and would benefit from a synopsis of the distributed architecture).

- The host used in testing

  It is assumed that the host configuration will not be a major factor in testing an embedded target's performance. Identification is shortened to identify the host and operating system:

  - the host hardware
  - the host operating system, including version
  - any special host characteristics which have a specific bearing on the execution environment

  Example:

  Host: DEC MicroVAX 3200 running VAX VMS Release 5.3-1.

- The tested target hardware

  The description contains two elements: 1) a high level description, providing a general description (e.g., a Motorola MC68020), and 2) a specific description of parts which would allow the reader to reconstruct the test hardware, assuming parts availability.

  The hardware description should be an inventory of all the independently configurable hardware elements that make up the tested system. The exact format of this section depends upon number of configurable elements in the system. The elements presented below are believed to be typical, but not exhaustive:

  - the processor type (e.g., 68020) [Generic]
  - the processor clock speed (e.g., 20 MHz) [Generic]
  - the memory cycle speed, (e.g., 60 milliseconds access time, no wait states) [Generic]
  - a list of the type and manufacturer of the target system (e.g., a Motorola Microsystems MVME141 board with an MVME 225 8 MB memory board in MMS MVME945 VME Bus Chassis) [Specific]
  - further hardware description, which characterizes the test equipment. This is optional. When included, it should describe standard features of the target hardware for readers unfamiliar with the specific systems under test (e.g., 8 MB cache, pipelined processor, etc.) [Specific]
  - any configurable or optional hardware features (e.g., an optional math coprocessor)
  - the strapping options used for the tested hardware [Specific]
  - the serial and revision numbers for the target system (often unavailable) [Specific]

  Example:

  Target: Motorola MVME141-1 microcomputer: 25MHz MC68030 CPU and
  25MHz MC68882 floating-point co-processor; 32Kb SRAM; 64Kb
  zero-wait-state SRAM cache; 256-byte on-chip instruction cache;
  256-byte on-chip data cache. The MVME141-1 board is paired

with a Motorola MVME244-2 8Mb DRAM memory module board
in a Motorola MVME945 VMEbus chassis.

- The tested compiler:
  - compiler manufacturer
  - version number (including version date if known)
  - compiler fixes beyond the standard release numbers
  - versions for components (where individual components are separately identified and versioned, e.g., compiler, linker, loader...)

Example:

Compiler: Verdix Ada Development System (VADS) VAX VMS to MC68020/30, Version 6.0.5(f), with kernel modified by BSY-2 contractor General Electric.

- The tested run time or operating system, if a separate product:
  - runtime system manufacturer
  - version number (including version date if known)
  - runtime system fixes beyond the standard release numbers
  - configuration options used in the tested system. Since this often involves describing all parameters used to generate a runtime system, this can be a reference to an appendix.
  - any special options set at run time

The test(s) run on the configuration and the options used in preparing for the test run (e.g., the compiler options such as optimization level) are not described here, but in the following observations. It is assumed that the same configuration will be used by multiple tests.

Observation 1:

Observations contain the results from a test or tests, organized around a single important idea. The observation is a detailed look at the evidence that was used to answer the question, and should present that evidence in detail. This does not mean that all data needs to be included, but rather that the important information is presented.

An observation normally applies only to a single configuration or is drawn from a single reference. However, this rule may be ignored when comparative information is appropriate to the explanation. The general concept is that a single observation highlights the measurement method(s) used for a specific data point used in answering the central question. For example, an observation might appropriately measure and describe of the amount of memory used to support runtime checking for the topic "How does performance change when all checks are turned off?" while information about the time used by runtime checking should be placed in another observation.

An observation opens with a statement of what was observed (or at least what was attempted), describes how it was observed, and discusses the meaning of the observations. The section then closes with raw data when appropriate.

Topics that should be addressed in an observation, in their normal order of presentation, include:

- the object of interest that the observation describes, such as the amount of time typical loop constructs require for execution
- the configuration(s) where the observation applies. This should reference the **Configuration(s) tested** section, and, optionally, may discuss if the observation can reasonably be extended to other systems
- the method(s) used to draw conclusions
- significant observations, the test(s) they were derived from, and their interpreted meanings
- the limitations of method or information which restrict the conclusions. The analyst should scrupulously draw attention to limits of the methods used and any missing data
- relevant raw data and options used to generate that data, in the format described below

The writer may vary the order of presentation for greater effect, but should address all the topics.

As mentioned above, the observation closes with the raw data. This output may be from a variety of sources, most commonly from the major Ada benchmark suites. An underlined, normal font title line should be used to distinguish between sources. The most common are included below, but additional entries should be made for other data sources.

Generic "Raw" Data and Source Paragraph Format

After opening the observation with the synthesis of the data, the observation section includes relevant raw data. The data are presented in a series of subsections divided by subsection headings (underlined text). The subsections designate the different data sources, most commonly tests from a benchmark suite (e.g., the Ada Compiler Evaluation Capability), but sometimes reports from other workers or published articles. Within each subsection, there will be one or more raw data elements containing one to three parts as separate paragraphs:

- An identifier for the source of the data which would allow the user to find and repeat the construction of the data.

  This identifier includes several parts:

  - The configuration tested. This is a back reference to the **Configuration(s) tested** section:

    Configuration 1

  - The test suite and version used, for example:

    PIWG Ver. 12/12/87

    ACEC Version 2.0

    AES Version DIYAES 2.0

    If the data was from published sources, the source should be cited in the references section and briefly noted here.

- The test or tests which generated the data (or, for some outputs the group of *related* tests):

    Test T000001 ☞ *An individual test*

    SSA Loop Variations report ☞ *Grouped tests*[1]

    Group I, Test TI10 ☞ *Group plus test id*[2]

- Include any modifications or options used to prepare tests that are *not* default values. For example, the default in the PIWG tests is to run tests with checking enabled. If tests are also run with checking disabled, this should be noted.

    PIWG Ver. 12/12/87, T000001A, Checks Off, Optimization Level 8

The identifier typically occupies a single line:

    Configuration 1, PIWG Ver. 12/12/87, Test T000001A.

However a multiple-line format should be adopted when a large number of individual tests are included:

    Configuration 1, ACEC Release 2.0, SSA Reports of "Language Feature Over-head":

    "Small Boolean Arrays (unpacked vs packed) =, AND, NOT"

    "Small Boolean Arrays (unpacked vs packed) =, AND"

    "Small Boolean Arrays (unpacked vs packed) /=, AND"

    "Large Boolean Arrays (unpacked vs packed) AND" [tests ss351, ss348].

    "Large Boolean Arrays (unpacked vs packed) AND" [tests ss350, ss353].

Care should be taken to ensure that the description uniquely identifies the source of test data.[3]

- (Optionally) textual information or graphical presentations. Particularly relevant would be explanations explaining anomalies in the data (e.g., erroneous observations, values out of expected ranges, etc.).

- (Optionally) some or all of the raw data. The output may be trimmed to eliminate unimportant details, but should not be rewritten or interpreted within this section. However, where results are erroneous or misleading, a note should be inserted to that effect.

---

1. In this case, the individual tests are grouped and listed in the raw data output and the test names need not be repeated individually.

2. The AES test names are unique, but knowing the next level in the hierarchy (the group) is often convenient.

3. The ACEC SSA (Single System Analysis) Report used for this document uses the same title for more than one section; these sections were distinguished by listing the individual tests contained within the section when necessary.

Included raw data should be clearly distinguished from analysis text by size, font or other means. For this report the OutputExample paragraph format was used which uses indention and a reduced size, fixed-width font.

While more than one test can be included within the raw data subsections, it is assumed that the number of entries will be relatively small. For now, no "sub-subsection" headings are contemplated.

Included below are examples of the heading for the most common benchmark suites used in preparing this report:

<u>PIWG Test Results:</u>

This heading is used to identify output from the PIWG test suite.

```
EXAMPLE OF OUTPUT FONT. Allows 80 characters per line.
```

<u>ACEC Test Results:</u>

This heading is used to identify output from the ACEC test suite.

<u>AES Test Results:</u>

This heading is used to identify output from the AES test suite.

References

- <entry here>

The references section lists all supporting documentation used in the entry. Citations should be complete enough to permit the reader to locate the resource. General format is:

AUTHOR(S); TITLE; {JOURNAL or BOOK or ORGANIZATION;} {VOLUME or PUBLISHER or RE-PORT #;} {PAGES}; DATE.

Each reference cited in the entry should be entered here, even if previously included in other entries. If no other references have been cited, a single line with the word "none" should be used:

- none

References

[ANSI]   ANSI; *The American National Standard for the Ada Programming Language*;
              American National Standards Institute, Inc.; 1430 Broadway, New York 10018;
              1983.

[BSI]   BSi Quality Assurance; Ada Evaluation Reader's Guide, Version 5.0; PO Box
              375, Milton Keynes, MK14 6LL, United Kingdom; 1989.

[BMAC]   Boeing Military Airplanes Company; Ada Compiler Evaluation Capability
              (ACEC) Technical Operating Report (TOR) Reader's Guide; Document Num-
              ber D500-12471-1, Release 2.0, April 24, 1990.

[GE]   General Electric Company: Software Standards and Procedures Manual for
              the AN/BSY-2 SUBMARINE COMBAT SYSTEM; 77C950014-B; Contract No.
              N00024-88-C-6150; CDRL Sequence No. B027-03-C1-002; D.L.D. No. DI-
              MCCR-800011; Rev. B; 28 September 1990.

[Ichbiah]   Ichbiah, Barnes, Firth, Woodger. *Rationale for the Design of the Ada Program-
              ming Language*. United States Government, 1986.

[Motorola]   Motorola; *MC68030 Enhanced 32-Bit Microprocessor User's Manual, Second
              Edition*; Prentice Hall, Englewood Cliffs, NJ 07632; 1989.

[Squire]   Squire, J.S. (editor), "Ada Numerics Standardization and Testing," *Ada Letters*,
              special edition, Vol. XI, No 7, Fall 1991 (I).

[NASA]   NASA SEAS (Systems, Engineering, and Analysis Support) Program Techni-
              cal Note. NASA Goddard Space Flight Center, Flight Dynamics Division /
              Code 552. April 10, 1990.

[VADScross-a] VADScross Verdix Ada Development System for Cross-Development Environ-
              ments, Version 6.05; VAX VMS ⇒ MC68020/30; User's Guide; Verdix Corpo-
              ration; January 16, 1991.

[VADScross-b] VADScross Verdix Ada Development System for Cross-Development Environ-
              ments, Version 6.05; VAX VMS ⇒ MC68020/30; Programmer's Guide; Verdix
              Corporation; January 16, 1991.

Index

P

parameters, number of
    procedure 88, 95
    rendezvous 169
Performance Issues Working Group (PIWG)
    benchmarks 3, 196
performance questions 181–193
    BSY-2 SSP Ada Style Guide questions 181
    combined list of questions 187
    final questions list 192
    format and instructions 197–204
    SEI additional questions 183
performance reports 3–5
pragma
    inline 63
    optimize_code 100
    passive 169
    suppress 7, 26
    volatile 100
precision 157–162
    extended 157
    single 157
private types 163
procedures
    generic 55
    inline 63
    location 88
    size 85–98

R

range_check 25
record 165–167
    aggregate 165
    location 31
rendezvous 169–180
    exception 53
reports
    performance 3–5
representation clause 43
run-time checking 7, 25–29
    array 21
    suppression 25

S

selective wait 175
size
    integer 160

    module 85–98
statements
    case 69
    exit 77
    goto 77
    if 69
    loop 77
    new 7
storage creep 38
storage_check 25

T

task
    priority 175
    rendezvous 169
types
    enumeration 43
    numeric 157–162
    private 163
    record 165–167
    visible 163

V

visible part 163

W

Whetstone benchmark 159

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
Unclassified	None

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
N/A	Approved for Public Release
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Distribution Unlimited
N/A	

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-92-TR-32	ESC-TR-92-032

6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION
Software Engineering Institute	SEI	SEI Joint Program Office

6c. ADDRESS (city, state, and zip code)	7b. ADDRESS (city, state, and zip code)
Carnegie Mellon University Pittsburgh PA 15213	ESC/AVS Hanscom Air Force Base, MA 01731

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
SEI Joint Program Office	ESC/AVS	F1962890C0003

8c. ADDRESS (city, state, and zip code))	10. SOURCE OF FUNDING NOS.			
Carnegie Mellon University Pittsburgh PA 15213	PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
	63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)

Performance and Ada Style for the AN/BSY-2 Submarine Combat System

12. PERSONAL AUTHOR(S)
Neal Altman, Patrick Donohoe

13a. TYPE OF REPORT	13b. TIME COVERED		14. DATE OF REPORT (year, month, day)	15. PAGE COUNT
Final	FROM	TO	December 1992	216

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Verdix Ada Development System (VADS), benchmarking, AN/BSY-2,
			performance analysis, Ada coding, BSY-2

19. ABSTRACT (continue on reverse if necessary and identify by block number)

The performance of programs prepared with the Verdix Ada Development System (VADS) was measured and analyzed for programmers preparing a large Ada system. Using standard Ada benchmark suites (ACEC, AES and PIWG) and a representative Motorola 68030 target system as a source of data, questions were posed and answered about programming alternatives, based on the measured performance of the compiler. The questions included in the report were abstracted from a much larger set selected from an analysis of the BSY-2 Style Guide and augmented with additional questions suggested by SEI experience. The derivation of the questions and the template for the performance analysis sections are presented as appendices.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■	Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (include area code)	22c. OFFICE SYMBOL
Tom Miller, Lt Col, USAF	(412) 268-7631	ESC/AVS (SEI)